

An Architecture for Insider Misuse Threat Prediction in IT Systems

by

Georgios V. Magklaras

This dissertation is submitted to the University of Plymouth

In fulfilment of the award of

MASTER OF PHILOSOPHY

School of Computing, Communications and Electronics

Faculty of Technology

January 2005

Abstract

An Architecture for Insider Misuse Threat Prediction in IT Systems

Georgios Vasilios Magklaras BSc (Hons)

The ever increasing computerization of business processes and mission critical applications, combined with the rising number of Internet technologies, has created new security threats for computer systems and networks. Numerous studies indicate that amongst the various types of security threats, the ones that originate from legitimate user actions can have serious consequences for the health of IT infrastructures. Although incidents of external origin are also dangerous, the insider IT misuse problem is difficult to address for a number of different reasons.

This thesis is concerned with the systematic study of the nature of Insider IT misuse problems, as well as the development of experimental insider IT misuse prediction techniques. The systematic study of legitimate user misuse actions is necessary due to the composite and variable nature of Insider IT misuse.

The thesis contains the results of a small scale survey that highlighted many important aspects of insider misuse actions. The results formed the basis for a suitable Insider Misuse Threat Prediction Factor Taxonomy, the end product of the systematic examination of the insider IT misuse phenomenon. The taxonomy was then used to construct a systems architecture that facilitates legitimate user threat prediction.

Although the proposed experimental architecture is far from the quality of a production-level utility, it constitutes a novel Insider Threat Prediction Model, which at the time of writing is unique in terms of its comprehensive design. It is considered that the predictive techniques could be taken forward in future research, in order to enhance the capability of existing Intrusion Detection Systems and aid IT professionals to mitigate Insider threats effectively. Various aspects of the proposed threat prediction model, the Insider IT misuse survey, as well as the proposed Threat Prediction Taxonomy have been published in conference proceedings and journals.

ACKNOWLEDGEMENTS

Completing a research degree whilst having a demanding full time job was a great challenge for me. This is the reason I am indebted to many people for their help as I wrote the thesis and playing with my little private computing laboratory.

First and foremost, I would like to thank my supervisor Dr. Steven Furnell, not only for his valuable editing and guidance role but also for encouraging me to start and (most importantly) finish my research degree. I am also grateful to the director of the Biotechnology Centre at the University of Oslo and current employer, Professor Kjetil Tasken, for allowing me to use the institution's IT infrastructure to conduct vital experiments. I should also not forget the Biotechnology Centre scientists for happily participating into my experiments and giving me permission to probe their user sessions.

My heartfelt thanks go also to Ida, who I admit I have neglected a lot, only in my attempt to cope with the demands of the job and the degree at the same time. Her calmness and patience reserves were always greater than my long hours of work and I will never forget that.

Lastly, it is hard for me to overstate the help I got from my parents. They never stopped re-assuring and helping me in every possible way they could, expecting nothing in return. Thank you!

Oslo, September 2004

Georgios V. Magklaras

AUTHOR'S DECLARATION

At no time during the registration for the degree of Master of Philosophy has the author been registered for any other University award.

This study was financed with part funding from the Engineering and Physical Sciences Research Council (EPSRC).

Relevant conferences and security events were attended during the course of the research. In addition, several papers were prepared for publication in refereed international journals and conferences, details of which are listed in the appendices.

Signed

Date

CHAPTER 1

INTRODUCTION

In our modern age, many people are enjoying the benefits of Information Technology (IT). The developed nations of the world are using Information Technology to transform the basis of their business transactions. Business critical infrastructures utilise IT infrastructures, in order to realise electronic commerce projects. E-commerce is currently one of the greatest ways to introduce a new and scalable global economy model. On the other hand, a plethora of critical infrastructures such as the telecommunications networks, air traffic control, energy and water distribution systems are also strongly dependent on computing platforms. Hence, the security of Information Technology infrastructures should be one of the most important considerations of system designers, operators and managers.

However, the term ‘computer security’ can be quite ambiguous and misleading, mainly because of its wide context. Even experienced computing professionals give different extensions to the term ‘Computer Security’. The basic notion and the extensions are discussed in detail in latter chapters of the thesis and they indicate the breadth of the different Information Security areas, as well as the extent of the problem. In fact, a great majority of the IT infrastructure components exhibit security flaws that render them susceptible to many forms of abuse. This is evident from a large number of Information security related surveys during the last four years, such as [1]. These surveys indicated a sharp rise in the number of security breaches that originated from external (i.e. unauthorised users) sources. The threat of an external penetration (often referred to as ‘hacking’) had been evident for years, but it started receiving widespread attention, especially from the mass media. Proprietary information theft from large enterprises, embarrassing web site defacements and devastating denial of service attacks forced the IT industry to launch a variety of security tools that help users and system administrators prevent, detect and -where possible- counteract IT abuse from external hackers. Computer anti-virus toolkits, firewalls, cryptographic software, Intrusion Detection Systems (IDS) and IT security policy shaping tools are the most common approaches followed by security experts today.

However the Information Security world has recently started becoming aware of another threat that had more devastating consequences and was substantially more difficult to tackle. This time, the threat was

not coming from external hackers, but from authorised users of IT systems. These users abuse their privileged access rights by committing a series of unintentional or deliberate actions damaging individuals or organisations in many different ways. The dissemination and storage of offensive material through e-mail and the stealing of proprietary information for rival companies or organisations are probably the most traditional cases of insider IT misuse known at the time of writing. Despite the well documented and emerging insider threat, there is currently no substantial effort devoted to addressing the problem of internal IT misuse. In fact, the great majority of misuse countermeasures address forms of abuse originating from external factors (i.e. the perceived threat from hackers).

1.1 Aims and Objectives

This thesis aims to investigate innovative approaches of dealing with authorised users that abuse IT systems. The overall aim is to advance the state of the art in the design and realisation of IDS by providing mechanisms to predict the level of threat that originates from legitimate users. The work results in the specification and evaluation of techniques that substantially extend the intrusion detection capability for IT system and network administrators and provide a significant enhancement to the overall security of IT systems. A number of specific objectives (of equal priority) apply:

Objective 1. To investigate the real nature and magnitude of the Insider IT Misuse problem by means of reviewing relevant information security surveys and devise a bespoke survey, taking into consideration the opinion of computing professionals.

Objective 2. To introduce a taxonomy of insider IT misuse incidents, in order to aid the process of modelling insider threat.

Objective 3. To propose a preliminary Insider Threat Prediction Model, that will profile legitimate users and estimate the level of threat for each individual legitimate user.

Objective 4. To derive an architectural framework for the development of a prototype Insider Threat Prediction Tool (ITPT) system, for building the proposed Insider Threat Prediction Model on a real world Operating System and test it against a number of selected Insider IT misuse scenarios.

1.2 Thesis Structure

Prior to examining the previously mentioned issues, it is essential that the reader becomes familiar with the latest advances in the field of IDS. Hence, the second chapter of this thesis examines in detail the notion of the term ‘computer security’ and provides an up-to-date overview of the currently employed IDS techniques. Special emphasis is given to considering the benefits as well as the weaknesses of each method and the mentioning of architectural frameworks whose contribution has advanced the field of Intrusion Detection research.

The discussion then moves to determine the magnitude of the Insider IT misuse problem (chapter three), a step that further refines the definition of the research problem domain. Some important field terminology is introduced, followed by an analysis of recent Information Security surveys, in order to qualify and quantify the nature of the Insider IT misuse problem.

Chapter 4 presents the results of an ‘Insider IT misuse survey’, one of the first systematic efforts to query different organisations specifically about the problem domain of this thesis. The rationale behind the design and distribution of the survey is explained and the derived conclusions direct the subsequent research and development steps of the project.

After discussing the insights of the Insider IT misuse problem domain and its magnitude, chapter five takes the research and development efforts one step further by introducing a comprehensive taxonomy of Insider Misuse Prediction Factors. Classifying IT abuse that originates from legitimate users is a vital step for systematising the research efforts in the area. In addition, the derived taxonomy lays the foundations for the development of the Insider Threat Prediction Model (ITPM), a mechanism that associates the likelihood of the occurrence of legitimate user IT abuse scenarios with certain system events. Chapter six discusses in detail this association by presenting the derivation of a suitable Insider Threat Prediction Model, which constitutes an additional novel area of this research. The selection of legitimate user monitoring criteria in order to profile authorised IT infrastructure users will be justified. The chosen criteria will then form the basis for a threat prediction function, a mechanism that associates certain user attributes to the likelihood of abusing the IT infrastructure.

Chapter seven integrates all the proposed techniques into an architectural framework that will realise them. The functional blocks of the Insider Threat Prediction Tool are presented. A group of current standards that will allow information exchange amongst the various system components is considered, followed by a justification of implementation criteria that are necessary for the development of a minimal pilot system for further experimentation.

The conclusions and limitations of the system are discussed in the eighth and final chapter of the thesis. The chapter discusses the performance of the system on a small number of Insider Misuse scenarios and concludes with recommendations for future work on a more rigorous system validation procedure, as well as future methods that can enhance the accuracy of the insider threat prediction process.

The appendices provide a plethora of detailed references to relevant technology standards, experiments as well as copies of publications associated with this research project.

CHAPTER 2

COMPUTER SECURITY AND INTRUSION DETECTION

This chapter explores fundamental issues that relate to the operational principles of Intrusion Detection Systems. These are important tools in the battle against computer security breaches and it should be clear that the ultimate goal of this research project is to enhance their capability in detecting and predicting threats that originate from authorised users. Hence, the first important step is to understand the basic philosophy behind their design and implementation. An overview of the history of IDS development is provided, followed by a critical evaluation of the major techniques used to intercept security breaches and a reference to important architectural Intrusion Detection frameworks.

2.1 The notion of the term ‘Computer Security’

Prior examining an IDS as a computer security tool, it is useful to clarify the term ‘computer security’. It is impossible to include an exhaustive plain English definition of the term. On the other hand, what we can say is that the wide context of the term divides its notion in two areas. The distinction between these two areas is quite fundamental when it comes to research and development methodologies for resolving computer security issues.

One area is related with formal methods that characterise security properties in a detailed and structured manner. The level of description may also justify mathematically a set of metrics and form security models that are provably correct. Bace and many other Information Security research scientists [2] mention the ‘security triad’ that places Information Security issues under three different headings:

Confidentiality is concerned with restricting access to information to only those users authorised for accessing the information. For instance, a public web server might contain a ‘members only’ area. The information contained in this area is restricted to only certain users and hence the server should provide the means for scrambling the information of the session of authorised users, so information in transit (through data networks or local Operating System processes) might not be viewed by unauthorised third parties. The application of cryptographic algorithms is a commonly accepted practice for enforcing confidentiality. Although cryptographic algorithms provide powerful means of preserving data confidentiality,

they cannot themselves prevent the occurrence of other conditions such as information alteration or deletion, as discussed in the following paragraphs.

Integrity is the requirement that information must be protected from intentional or accidental alteration. Using the previous example, our public server should contain tools that prevent malicious alterations of the web page content. As an example, nearly all operating systems contain a filesystem mechanism that verifies all actions (create, delete or alter files) against a particular user identity. This process is commonly referred to as ‘access control’ and there are several different mechanisms to achieve this goal, each with a varying degree of reliability. It is outside the scope of this thesis to provide a detailed coverage of cryptographic and access control algorithms. A concise overview of these issues is provided by Phoenix [3], as well as Skevington and Hart [4].

Availability is the final element of the computer security triad and represents the requirement to have an IT infrastructure with system resources that are able to continue to work under a variety of scenarios. This means that authorised users of the IT infrastructure are able to access resources when and where they need them. The replication of the data content as well as the physical resources of an IT infrastructure for redundancy purposes are a good example of availability related measures. For instance, the contents of the hard drive of a web server, its network link or even the entire server might be replicated to counteract accidental (flood, earthquake) or intentional (arson, theft, Denial of Service attack) damage.

In addition to the previously mentioned classic triad of secure system properties, someone should also emphasize the property of *accountability*. A system is said to exhibit accountability properties if it is capable of *reliably* associating a given activity to the party responsible for the initiation of this activity. The term ‘reliably’ refers mainly to the ability of the system to provide an unaltered record of these associations and it has certainly a lot of common ground with the property of integrity. However, collecting the right type and amount of accountability data on large IT infrastructures is a non-trivial task that goes far beyond the task of maintaining their integrity [5].

In theory, if an IT infrastructure satisfies all the previously mentioned requirements, it can be deemed as a secure one. Moreover, a plethora of technologies try to satisfy one or more elements of the triad. Internet and Intranet firewalls, for example, attempt to protect both the confidentiality and integrity of important information. Zwicky, Cooper and Chapman [6] define a firewall system as a single network traffic choke point, which prevents the security dangers of the Internet from spreading to your internal network. Figure 2.1 illustrates a typical firewall application scenario.

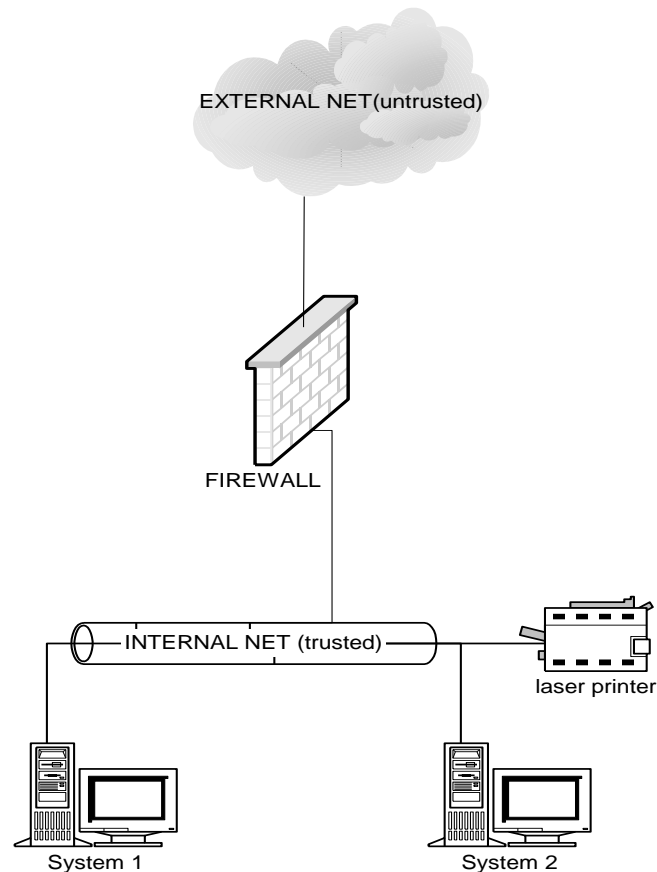


Figure 2.1: A basic network fire-walled from the Internet

The firewall host stands always between the non-trusted network (Internet) and the internal internetwork. Its role is to prevent access of unauthorised individuals into the internal network. This is possible by authenticating incoming and outgoing packets against the source, destination endpoint address and port number (packet filtering firewalls). It is also possible to be more sophisticated and grant access to network traffic by checking the payload and appearance sequence of each Protocol Data Unit against a list of malicious payloads and sequences (stateful firewall). However, firewalls can impose dramatic limitations on the performance of large networks, they are difficult to configure and

they deal mostly with external threats. Their application is a preferred feature for network security but not a panacea.

The other way to clarify the fuzzy term ‘computer security’ is by considering a practitioner’s approach and formulate a generic, neat perception of the overall Computer Security domain. Garfinkel and Spafford [7] adopt that kind of practical view and support the opinion that “a computer is secure if you can depend on it and its software to behave as you expect.” This is a less formal definition of the term, based mainly on ‘hands-on’ experience of technical computing issues.

If someone poses the question of which definition should be adopted for the purposes of research and development, the answer would point to the first and formal one. This is not to say that Garfinkel and Spafford are on the wrong track. The earlier definition is more suitable for the formal research environment because it provides more systematic and quantifiable security evaluation criteria. Expected behaviour is not an objective criterion. Someone could argue the fact that there are different expectations for the behaviour of a computer system between a software engineer and a line manager, since each of them might have a different list of desirable features (the manager might disagree with the software engineer and favour confidentiality over availability). This level of ambiguity can confuse the security evaluation process and it will certainly always require additional and more formal clarifications about what is considered as expected behaviour for the system.

Whatever the definition, there is one universal truth about computing infrastructures. They always exhibit important design flaws that render them susceptible to many different kinds of security breaches. Denning’s seminal work on IDS [8] points out that despite the widespread deployment of cryptographic, authentication and firewalling technologies, weaknesses that reside in software applications, Operating Systems and the level of knowledge of technical staff will always open the door to malicious abuse of computing systems. Hence, if it is not possible to prevent the occurrence of these events, it would be at least useful to know when, how and from where these malicious acts originate. The tools that arm the IT specialist with information to adequately answer the previously mentioned questions are collectively called ‘Intrusion Detection Systems’ (IDS). The remaining chapter sections examine their history and principles of development.

2.2 The birth of the Intrusion Detection System

The concept of Intrusion Detection evolved from the notion of automated audit trail processing by Anderson [9]. The basic idea behind an audit trail is the careful selection of indicators that reveal important events about the status of a computing system or the actions that bring the system to a particular state. Each indicator is called an ‘audit probe’ and is usually selected by the security administrators that look after the computing platform. Figure 2.2 below illustrates the concept.

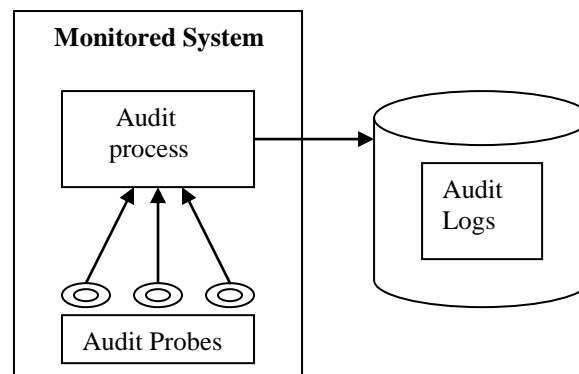


Figure 2.2: The principle of audit log generation

The ‘Monitored System’ has a set of active audit probes, shown as rings. These probes communicate with a software audit process. The software process then updates a file, which essentially is an archive of all the events intercepted by the probes. Information is usually stored by using the ASCII or UNICODE character set, although exceptions do exist and create audit archives in proprietary binary forms.

The generation of audit logs was (and still is) an important requirement for the security of computing platforms. In 1987, the American National Computer Security Centre published the “Guide to Understanding Audit in Trusted Systems” document [10]. The document tried to interpret the complex audit requirements of the famous United States Department of Defence ‘Orange book’ [11]. Amongst other things, it made clear that every Operating System should provide audit-logging facilities. In fact, the security compliance level of an Operating System was directly proportional to the wealth and reliability of its audit probes. Hence, it suggested the primary technical and administrative goals of an Operating System audit mechanism, as well as a list of preferred auditable events.

A traditional example of an audit log generation process is the ‘syslogd’, the standard error logging process employed by many UNIX-like Operating Systems, originally written by Eric Allman at the University of Berkeley. Garfinkel and Spafford [7] provide more details about the setup and utilisation of this logging utility. However, we provide a small snapshot [Figure 2.3] of what a syslog audit file looks under the LINUX operating system. Amongst various recorded events, there are important indications (bolded) that someone has repeatedly tried to gain unauthorised access to the server.

```
Feb 16 19:25:15 archimedes kernel: parport0: PC-style at 0x378 [PCSPP,TRISTATE] Feb 16 19:25:15 archimedes
kernel: lp0: using parport0 (polling).
Feb 16 19:25:15 archimedes lpd: lpd startup succeeded
Feb 16 19:25:16 archimedes gpm: gpm startup succeeded
Feb 16 19:25:16 archimedes crond: crond startup succeeded
Feb 16 19:25:18 archimedes xfs: xfs startup succeeded
Feb 16 19:25:18 archimedes xfs: listening on port 7100
Feb 16 19:43:49 archimedes login(pam_unix)[1186]: authentication failure; logname= uid=0 euid=0 tty=pts/2 ruser=
rhost=192.101.101.103 user=gmagklas
Feb 16 19:25:18 archimedes anacron: anacron startup succeeded
Feb 16 19:25:26 archimedes login(pam_unix)[870]: session opened for user gmagklas by (uid=0)
Feb 16 19:25:27 archimedes login[1186]: FAILED LOGIN 1 FROM 192.101.101.103 FOR gmagklas, Authentication
failure
Feb 16 19:25:27 archimedes login(pam_unix)[870]: session opened for user gmagklas by (uid=0)
Feb 16 19:25:28 archimedes -- gmagklas[870]: LOGIN ON pts/0 BY gmagklas FROM 192.101.101.103
Feb 16 19:25:48 archimedes su(pam_unix)[954]: session opened for user root by gmagklas(uid=500)
Feb 16 19:43:59 archimedes login[1186]: FAILED LOGIN 2 FROM 192.101.101.103 FOR gmagklas, Authentication
failure
Feb 16 19:44:06 archimedes login[1186]: FAILED LOGIN 3 FROM 192.101.101.103 FOR gmagklas, Authentication
failure
Feb 16 19:44:18 archimedes login(pam_unix)[1186]: service(login) ignoring max retries; 4 > 3
Feb 16 19:45:43 archimedes su(pam_unix)[954]: session closed for user root
Feb 16 19:45:48 archimedes su(pam_unix)[1191]: session opened for user root by gmagklas(uid=500)
```

Figure 2.3: Snapshot of a ‘syslogd’ generated log

The important point to consider here is that the entries that indicate attempts to breach system security (indicated in bold letters) are very few when compared to the overall number of audit record entries (one per line). In the early days, the security administrator would have to manually parse each audit record individually, decide what was relevant, discard the rest and take appropriate actions. Large mainframe systems (with thousand of users and many complex applications) with a notable number of audit probes would generate several thousands of audit records on a daily basis, making the manual process of extracting relevant information extremely tedious.

Anderson [9] published the ‘Reference Monitor’ concept in a project funded by the United States Air Force to address the problem of filtering important information out of enormous log files. The Reference Monitor was a mechanism that eliminated automatically redundant or irrelevant records from security audit trails. This is formally called ‘audit reduction’. Its application had a profound

impact on computer audit mechanisms and was a tool that substantially reduced the load burden and increased the efficiency of security administrators.

The automatic audit record processing had set the foundations for the IDS concept and in 1985, a research group founded by the United States Navy Command created the ‘Automated Audit Analysis’ system [12]. The prototype utilised data collected from the shell environment of a UNIX Operating System. The data was then analysed by using Relational Database Management tools and the research pioneered a way to distinguish normal from irregular system usage.

The study of irregular system usage was the subject of another United States Navy Research team. From 1984 to 1986, Dorothy Denning and Peter Neumann were the first to introduce the term Intrusion Detection. They researched and developed a model that proposed a correlation between unusual activity and misuse. Their project was eventually named ‘the Intrusion Detection Expert System (IDES)’ and formed the basis for the seminal ‘Intrusion Detection Model’ [8] paper, published in 1987 by Dorothy Denning. Teresa Lunt (of the ‘Automated Audit Analysis’ project) joined the previously mentioned pioneers and continued working on the IDES architectural framework. A prototype system was developed in a proprietary TOPS-20 computing platform. The work was finalised in the early nineties and the first results were published during the Sixth Annual Computer Security Applications Conference in Tucson [13].

The influence that the IDES project had on the computer security research world was phenomenal. The results created interest amongst various research teams around the globe and launched a large number of relevant projects. It was clear that the IDS concept was becoming very promising and the next section will examine in detail their principles of operation as well as their most important architectural frameworks.

2.3 The anatomy of an Intrusion Detection System

Computer Intrusion Detection Systems provide search functions, as well as the functionality to alert the responsible parties when activities of interest occur. As a consequence, the term IDS and the notion of the word intrusion are going to be used throughout this thesis only with reference to the ‘Information

Technology (IT) infrastructure'. The latter term refers to an organisation's set of discrete computer systems (dedicated servers, client workstations) and the telecommunications components that interconnect them, in order to perform a useful task.

In simple terms, Intrusion Detection is a vital technology component of a modern security management system. Its basic task is not only to prevent and (where possible) respond to a plethora of computer security incidents, but also to integrate the operation of other security components (anti-virus, firewall and cryptographic applications) into one all-rounded system. An IDS is a tool that monitors the events occurring in a computer system, searching for indications of security related problems.

However, a fundamental step in understanding the concepts behind Intrusion Detection is to clarify the term 'intrusion'. One of the most compact but yet descriptive definitions is given by Amoroso [14]. He defines an intrusion as *"a sequence of related actions by a malicious adversary that results in the occurrence of unauthorised security threats to a target computing or networking domain"*. Consequently, he defines Intrusion Detection as *"...the process of identifying and responding to malicious activity targeted at computing and networking resources"*.

Amoroso emphasises the term 'process', stating that it is a critical property of Intrusion Detection. This property involves a certain level of interaction between the technological tools that perform the actual detection and the people that administer or trigger them. It is this interaction that presents great challenges to the IDS administrator and constitutes a problem for Insider Misuse, as explained in later chapters. Amoroso also elaborates on the term 'malicious activities' by referring to security-relevant actions that are intentional. Although it could be argued that most external security breaches are intentional, the balance between accidental and intentional security breaches becomes unclear when someone considers threats from legitimate users, as discussed in the third chapter of the thesis.

The definition of what is considered as malicious or intrusive activity is also environment specific. For example, certain organisations such as government departments would consider as intrusive any activity that would result in unauthorised disclosure of sensitive information. On the other hand, the

unauthorised alteration of the contents of a web server might be the major concern for other establishments, such as news agencies and political parties.

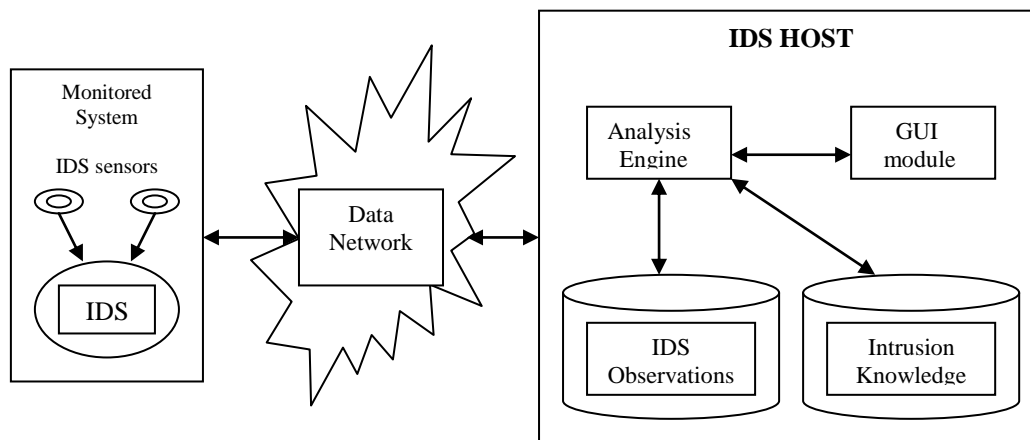


Figure 2.4: The functional blocks of a basic IDS

Based on the previously mentioned principles, figure 2.4 depicts the functional blocks of a typical IDS. The sensors constitute modules for collecting a predefined set of events from monitored systems. Repeated unsuccessful login attempts, modification or access of certain system files are some of the most typical examples of collected events.

The event logs are forwarded through the data networking infrastructure to a dedicated computer host that runs the IDS software (IDS host). The interpreted observations are stored into one or many file buffers that constitute the log archive of the IDS host. The knowledge base file buffer is a collection of useful information about what constitutes intrusive activity. This type of information might be predefined by a system specialist or intelligently inferred by the IDS itself, depending on what type of intrusion algorithms are employed. An important research issue in IDS knowledge bases involves the development of efficient and commonly accepted ways for encoding intrusion attack information [15].

At the heart of the system lies the ‘analysis engine’ that is responsible for running the Intrusion Detection algorithms. Whilst section 2.4 will contribute the necessary descriptions of various Intrusion Detection processing schemes, it is worth mentioning that the primary goal of these schemes is the identification of key intrusion evidence and the decision making about the initiation of certain types of responses.

Although automated responses represent notable IDS design trends [16], extreme caution is needed, in order to minimise the risk of the automated response being used as a vehicle for attack. A knowledgeable malicious intruder that has compromised a user account knows that he/she will probably be detected. If the hacker knows that the IDS might disable someone's account, he/she can launch a Denial of Service (DoS) attack. This can impact the management of the IT infrastructure, because it will certainly require manual intervention to restore the affected user account(s) and systems. Interested readers should refer to Chapter 12 of Bace [2], which presents evidence of the occurrence of real world cases where an automated response was exploited by malicious intruders. The issue of IDS response is also a major research and development issue on its own.

Finally, all the previously mentioned components are co-ordinated by means of a Management System that provides an intuitive Graphical User Interface (GUI). This is necessary, in order to provide an interface to the human operator. A key issue in IDS GUI design is the careful definition of what type of information should be displayed. Currently, there is no substantial experience for determining a commonly accepted way of displaying intrusion related information as mentioned in pages 27-28 of Amoroso [14].

Before the presentation of major IDS algorithms, it is good time to emphasise two important aspects of an IT intrusion. One of them concerns the temporal nature of computer intrusions. Earlier paragraphs stated that intrusions are sequences of related (i.e. intrusion-relevant) actions. This constitutes the basis for constructing a temporal model of computer intrusions, as indicated by part A of Figure 2.5. Time is indeed an important element in intrusion detection. An intruder usually begins with some initial action as the first element of the time sequence. This early step usually corresponds to an attempt to breach a security feature of the target computing system. Several intermediate actions might be logged and usually the final one indicates either a successful or flawed attempt to bypass the defence mechanisms of the system. In the event of the attempts being successful, a 'security effect' has taken place, indicating the violation of an anti-intrusion mechanism of the system. Subsequent actions might then follow that could potentially result on the occurrence of additional security effects.

However, part A of Figure 2.5 presents a rather simplistic view of the Intrusion Detection System problem. In practice, the hardest problem is differentiating between actions that are relevant to intrusive activities and actions that have nothing to do with attempts to bypass the security mechanisms of a system. Figure 2.3 demonstrated the problem by providing a plethora of audit records. A small number of them were relevant indicating unsuccessful TELNET attempts to the server. The TELNET attempts occurred at irregular time intervals. The rest of the audit record entries are irrelevant and present a special kind of ‘noise’ to the Intrusion Detection process. Hence, a more realistic intrusion temporal model is given in part B of Figure 2.5.

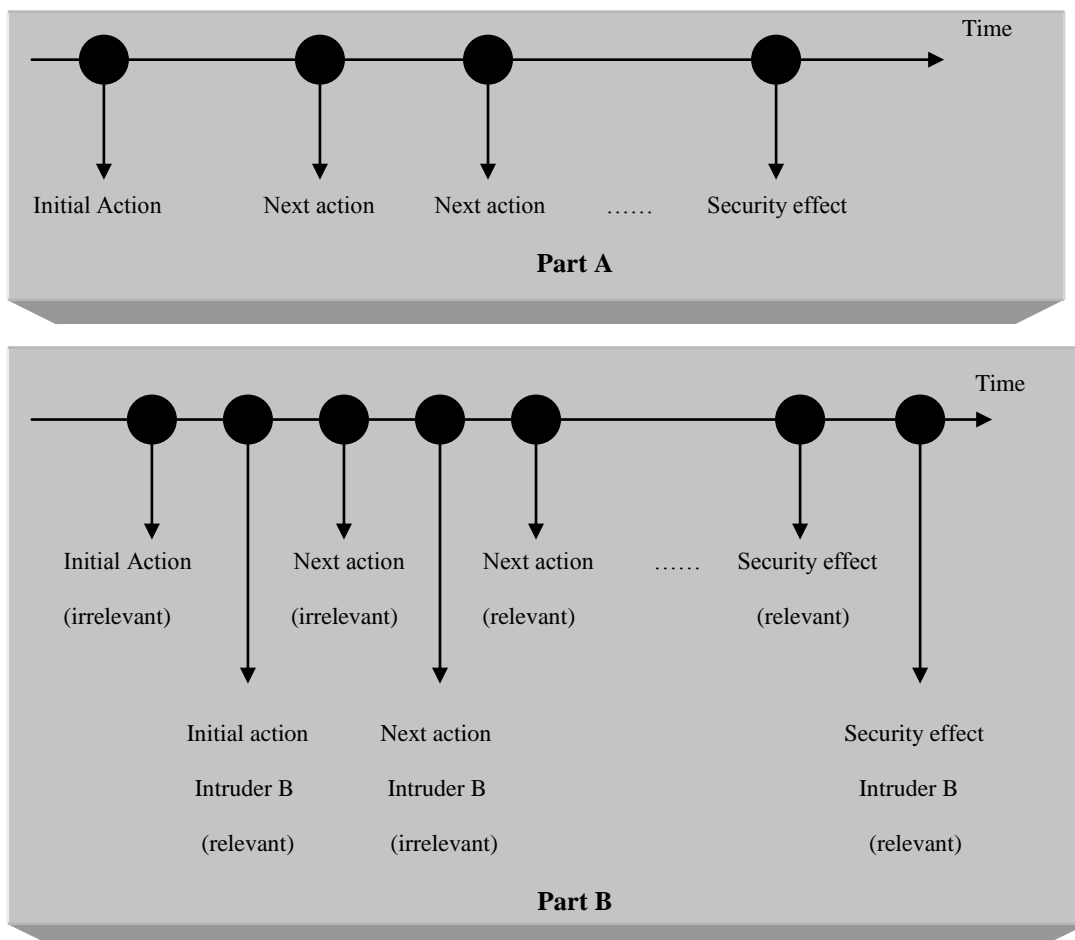


Figure 2.5: Temporal modeling of computer intrusions

Part B also displays actions related to more than one intrusive action. Most Intrusion Detection Systems examine each one of them separately by the establishment of ‘intrusion sessions’. Each session contains a list of targeted resources (hosts, applications, authentication mechanisms) associated

to a list of attack origins that usually include other hosts and potential user identities. The establishment of these lists represents another important issue in the Intrusion Detection process, that of accountability, as earlier discussions point out.

Accountability or ‘event traceability’ mechanisms present one of the greatest challenges of Intrusion Detection Systems research. It is not always possible to trace back the point of origin of certain attacks. Many factors can be considered that make it possible to complicate or even hide identity in computing infrastructures. The inherent insecurity of the TCP/IP protocol (alteration of source IP address – IP spoofing), the use of cryptographic protocols to scramble the content of IP packets and the inadequate configuration of network devices (routers and firewalls) are some of the most important techniques that can be exploited by potential intruders. It is outside the scope of this thesis to analyse all the potential ways of achieving this goal. However, the reader can refer to Staniford and Heberlein [17] for a detailed reference of the previously mentioned issues.

2.4 Principles of Intrusion Detection techniques and architectures

Having discussed the basic elements of an Intrusion Detection System, this section relates to the techniques that perform the actual Intrusion Detection process. For each technique, we discuss the relative advantages and disadvantages and important architectural frameworks that employ them, in order to promote the research and development efforts of the Intrusion Detection field.

The reader will find many examples of relevant literature dividing Intrusion Detection Systems into host and network-based ones. A **host-based IDS** performs all the necessary computations by considering data that are sampled from the operating system and the applications that run on top of it. On the other hand, a **network-based IDS** considers sensor data that originate from the infrastructure that interconnects computer devices. Hence, a fundamental difference between the two is that the earlier might utilise data networks to disseminate information amongst the various IDS components, whereas the later seeks intrusive activity inside the core of a data network.

A Protocol Data Unit (PDU) is the fundamental building block for data network based communications and the fundamental source of information for a network based IDS. Figure 2.6 depicts a simplified

view of an Internet Protocol PDU that carries part of a UNIX /etc/passwd file from host 192.168.2.1 to host 192.168.2.2. No matter what the underlying network protocol might be, a PDU will always contain a header and a payload area. The header usually provides pointers to the origin and destination of the PDU, whereas the payload area encapsulates the actual information carried by it. A network-based IDS will contain special mechanisms to intercept protocol data units and copy certain parts of their header and payload areas to a memory buffer for further inspection and processing. A well-known architectural example of a network-based IDS product is the Network Flight Recorder [18], invented in 1997 by Marcus Ranum and other researchers.

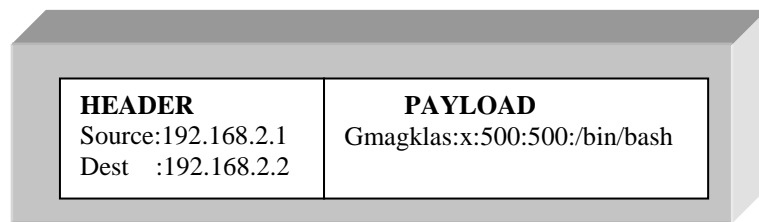


Figure 2.6: A Protocol Data Unit (PDU)

The two previously described IDS categories are not mutually exclusive. In fact, if someone considers the widespread usage of computer networks, it will become clear that the combination of host and network-based intrusion detection is a necessary strategy for devising an effective IDS. As a result, the border between network and host-based Intrusion Detection is currently more vague than ever, with most research frameworks and commercial products seamlessly integrating these two methods into one single system.

However, there are two issues that will affect the future of network-based intrusion detection. The increasing speed of Local and Wide Area (LAN/WAN) network technologies creates a scalability issue. Today, many LAN topologies operate on a speed of 100 Mbps, dictated by the IEEE 802.3u Fast Ethernet technology. Let us assume that a network based IDS is set up to intercept packets, listening to all LAN segments. If someone considers an average size of an Ethernet based frame of 800 bytes and assuming that the network operates at three quarters of its maximum capacity (75 Mbps), there will be on average nearly 12000 PDUs that hit each of its network interfaces every second. Extracting information from each one of the intercepted PDUs and performing the necessary computations to update operational values will induce a serious processing load for the CPU of the IDS. With LAN

backbone speeds of 1 Gigabit per second and over, the performance of sophisticated packet inspection engines will be under question, when it comes to rapid response times.

The second threat to network intrusion detection is the widespread deployment of cryptographic technologies that encrypt the payload area (and possibly the header) of a PDU. In an alarming article that comments on several aspects of network security, Bruce Schneier [19] describes how IPsec can degrade the effectiveness of a network-based IDS. Although it is true that encrypting network traffic can only complicate things when it comes to the interception of intrusive activities, the use of encryption to defend the privacy and integrity of messages is also necessary. Hence, an important task for a network designer or security architect is to find the right balance between encrypted and plaintext traffic, by identifying the network points that should utilise encryption and where traffic could be unencrypted for the purposes of efficiency and monitoring.

At the time of writing, there are many different Intrusion Detection algorithms under development. Moreover, it does not matter where the algorithms are applied at network or host level. Although the sensor probe technology is different between network and host based Intrusion Detection, the principles of computations are the same and all techniques can be categorised in two major schools of thought. ‘**Anomaly detection**’ is one major category of Intrusion Detection techniques, which intercepts intrusive activities by analysing statistical profiles of user behaviour over time. These profiles could also be used to monitor the behaviour of automated system processes that execute programs by means of a specific user identity. The second major method of Intrusion Detection tends to analyse intrusive events that are described in terms of rules and pattern descriptors. This technique is called ‘**misuse detection**’. These techniques will now be considered in the subsections that follow.

2.4.1 Identifying intrusive activity by using anomaly detection

Anomaly detection was one of the earliest approaches employed in Intrusion Detection architectural frameworks. In 1986, Denning’s Intrusion Detection Model [8] emphasised a very important observation. An intrusive activity often manifests itself as an unusual (i.e. abnormal) event that could be spotted by using a variety of statistical methods. This means that for a specific system and operation, it is possible to establish a profile of normal activity. This is done by carefully defining a set of metrics

that are indicative of intrusive activity and then perform a series of statistical calculations, in order to infer whether a user (or process) has irregular and hence suspicious behaviour.

Each metric is assigned a variable. The key notion of a mathematical variable is that it can be associated with a distinct value from a well-defined domain. In this particular case, certain variables might represent the amount of network connections of a user, the CPU usage, failed login attempts and many other intrusion related criteria, at a particular point in time. These values are usually cumulative (they are stored in arrays) and are regularly sampled over a pre-defined time interval. This interval can be fixed in time (set to zero at a particular hour of the day) or function over a sliding time window.

When an adequate number of samples has been collected, the values are fed to a statistical function. The arithmetic mean and the standard deviation (sd) functions are some of the simplest examples, whereas Markov chains and other types of stochastic processes might be included. The end result is the production of a set of permissible values (**thresholds**) for every variable that represents a metric. If the value of the variable is outside the pre-defined range, a threshold alarm will be triggered and the system will classify the event (or series of events) that produced that value as intrusive. A good example of a commonly used threshold is the number of permissible unsuccessful login attempts to a system, as previously demonstrated in Figure 2.3.

```
For every metric: sample set[n]=getval( n, probe);  
Metric value= sd(sample set[n]);  
Result = Compare (Metric value, Metric Threshold);  
If (result==true) {log("Normal result"); exit();} else  
{log("Abnormal result"); response(metric); exit();}
```

Figure 2.7: The principle of an anomaly detection algorithm

Hence, a generic procedural pseudo algorithm for a very simplistic anomaly detection system is illustrated above (Figure 2.7). A ‘sample set’ is a group of collected intrusion metric values. These values are usually stored in contiguous areas of memory cells and they are then fed to the anomaly detection based function that evaluates the mean and standard deviation. The sample size and hence the size of the array is indicated by an integer n. The calculated values will then be compared against a set of carefully chosen thresholds for each metric by the ‘compare’ function. The function returns true if there is no substantial level of variance between the thresholds and the derived values and false

otherwise. In the later case, an anomaly has been detected and that will usually force the system to respond according to a pre-defined procedure associated with each metric.

The previously mentioned algorithm is the same in principle for a wide variety of monitoring situations. It could be applied to building a profile of normal system or network operations. A subtle point in the anomaly detection process is the selection of suitable threshold values to distinguish between anomalous and normal activity. Certain users, applications or network traffic trends will change over time. If the IDS designer does not compensate for this feature, there will be false positive or negative alarms that reduce the accuracy of the anomaly detection process. A number of techniques intended to refine the threshold values are discussed in the following paragraphs.

Time series analysis was proposed by Denning [8], in order to dynamically adapt statistical profiles that change over time and may be abused by an attacker to gradually train the profile and thereby avoid the mechanisms of anomaly detection. The time series takes into account the order and inter-arrival times of the observations, making use of the temporary model of an intrusion as stated in earlier sections. An observation is abnormal if the probability of occurring at a specified time interval is too low or too high. This analysis model has produced accurate detection results. Its main disadvantage is that it requires vast amounts of computational resources (CPU and memory) and thus it does not scale.

The beginning of the nineties decade saw an explosive growth of the statistical anomaly detection research efforts. Predictive pattern generation [20] is another interesting anomaly detection technique that utilises the axioms of conditional probability, in order to predict future scenarios based on the events that have already occurred. It is highly adaptive to profile changes and uses a dynamic set of rules for detecting intrusions. The rules are not static. Instead, they are inductively generated based on the sequential relationships and temporal properties of the observed events. The identification of regular patterns of events allows the prediction generation algorithm to infer that some specified event types are more likely to occur next in the series of events than others. The algorithm assigns a probability to each most likely event. It then refines the assigned probabilities by inductively generating rules in the following form: **Consider an input sequence of events E_1, \dots, E_k . Then the rule for that specific sequence of events is: - $(E_{k+1}, P(E_{k+1})), \dots, (E_n, P(E_n))$** . The rule expression

could be interpreted in plain English: “Assuming that the input stream contains the event sequence E_1, \dots, E_k , the events E_{k+1}, \dots, E_n are the more likely to be seen in the rest of the input sequence, with corresponding probabilities of $P(E_{k+1}), \dots, P(E_n)$.”[20]

Predictive pattern generation has the advantage of focusing on a few relevant security events rather than the entire monitored session and can therefore be efficient in terms of computational resources. In addition, it has good tolerance to intentional training by malicious intruders who are trying to avoid detection. However, it has one major drawback. Its effectiveness is totally dependent on training the system by using well thought scenarios of abnormal activity and usually requires expert knowledge. If the inductively generated rules are not comprehensive enough to cover all possible abnormal events, certain events will be not flagged as intrusive (i.e. false negatives). A partial solution to this is to implicitly characterise every unknown event as anomalous, which has also the potential of introducing false positive alarms.

In 1995, Kumar [21] introduced Neural Networks as one of the latest strategies to aid in statistical profile adaptation. The basic idea is the training of the neural network on a set of representative user, application or network traffic characteristics that can certainly indicate abnormal activity. After the initial training period, the neural network receives activity data and determines to what extent the sampled activities exhibit similarities with the training samples. Abnormal data yields a notable change in the state of neural units, connections, or weights, flagging anomalous activity. However, the level of profile adaptation on a neural net is substantially greater than the time series equivalent methods.

Furthermore, a neural network has a relatively low impact on computational resources because it does not make prior assumptions on the expected statistical distribution of measures. It is more flexible than the rest of anomaly detection measures, because it does not employ a fixed set of metrics. However, this flexibility has a cost when it comes to detecting faults in their training. When a neural network detects an anomalous event, it will adapt its notion of normality by initiating a series of stepwise weight corrections. Tracing the reason for a detected anomaly through stepwise weight correction can be almost impossible. For this reason, the current state of the art in anomaly detection does not consider neural networks as a pure statistical detection method but simply as a valuable complement.

Finally, the latest anomaly detection technique inspired from the Artificial Intelligence computing field was presented in 1998 by Ludovic Me [22] and utilises **evolutionary computing algorithms** to perform the analysis of the collected data. The devised system 'GASSATA' defines hypothesis vectors from event data. The vectors either indicate an intrusion or not, making an initial hypothesis. They are then fed to a binary encoding function that represents them as series of binary digits (bits). A 'fitness function' accepts the binary coded vectors, randomly mutates selected bits and tests the validity of the newly produced individuals against a set of criteria, until an optimal hypothesis is devised. The results of this method are encouraging. The authors reported that the mean probability of true positives was 0.996, for analysing 200 user attacks in ten minutes and twenty five seconds. Clearly, there is going to be a lot of overlap between the fields of Intrusion Detection and Evolutionary Computation.

2.4.2 Identifying intrusive activity by misuse detection

The second major school of thought in Intrusion Detection tries to intercept intrusive activities by comparing audit probe data to a repository of attack descriptions or '**signatures**'. These signatures conform to a scheme that enforces a well-defined byte sequence describing intrusive activities. They normally reside on a plain file. The file is then consulted by an IDS on startup and constitutes its attack knowledge repository. The most common example of a misuse detection system that is employed widely in the commercial world is that of a computer anti-virus application [23]. Nearly all commercially employed anti-virus software packages use virus description files. These are carefully devised byte sequences that describe unique characteristics of malicious code in a bespoke (often proprietary) description language.

The major difference of this method with respect to anomaly detection is that intrusion knowledge is not made of threshold values produced by statistical calculations. Instead, the search activity is governed to a large extent by a direct comparison of the byte sequences of the signature file and those derived by the IDS sensors. As a result, misuse detection is more static than statistical based approaches and there is no efficient way of dynamically refining a misuse detection signature.

There are anti-virus packages that are able to intercept malicious code that has not been identified before. However, this is still based on pattern matching heuristic algorithms that are designed to intercept common actions during the execution of malicious code. An example is an executable program that tries to insert itself at the beginning or the end of certain files [23]. These techniques are an important feature that broadens the horizons of a misuse detection system, but they can miss features that have not been characterised as common actions of malicious code. This is the reason that all anti-virus vendors suggest frequent updates of the virus description files. Consequently, today there is not a known misuse detection method that can successfully detect an entirely new method of IT intrusion. One can only improve them to be effective against variations of existing attack methods or keep updating the attack signature repository, so that new threats can be addressed.

It is also important to note that misuse detection signatures might be characterized as ‘atomic’ or ‘composite’, depending on whether they describe aspects of a single event or they tend to codify characteristics that are spread across many events. An example of an atomic signature is one that detects a badly formed PDU, such as one used in the ‘land’ network attack [6]. This might cause the victim’s machine to crash on the reception of the packet. On the other hand, a signature that describes a port-scanning incident is considered a composite one, simply because the monitoring aspects need to maintain information that concerns many different types of packets, at different time intervals.

The invention of a suitable structure for storing signatures in a standardised and efficient way is an important issue in the research and development of misuse detection oriented algorithms. Efficiency is all about describing events in a compact (memory and algorithmic complexity) but yet unambiguous way. It also enables misuse detection systems to perform ‘on-the-fly processing’, where the IDS is able to initiate response activities in a more timely fashion, instead of doing ‘after-the-event’ analysis of audit records in batch mode. In contrast, anomaly detection is unable to perform these functions, mainly due to its computational complexity.

Standardisation is also a desired property of an attack description structure, because it can provide a quick way of disseminating attack descriptions amongst different IDS vendors and promote a fast

response to attacks. Unfortunately, most commercial IDS vendors keep their systems proprietary and normally attack descriptions from one vendor cannot be utilised by the products of another one.

One way of structuring the storage of attack signatures is a '**production**' or '**expert system**'. Expert systems consist of a knowledge base containing descriptions of suspicious behaviour. The description is based on rules that format the sampled data and perform an if-then style comparison, associating collected data with predefined knowledge. The 'if' part of the rule describes a matching condition that is formally defined by the methods discussed in the following paragraphs of this section.

One heavily utilised signature description structure can be produced by pattern matching engines. In the great majority of the cases, a pattern matching engine will apply a string matching algorithm. A string is a sequence of characters represented by either an ASCII or UNICODE character set. In the classic pattern matching problem on strings, an algorithm is given a text string T of length n and a pattern string P of length m . The algorithm aims to find whether P is a sub-string of T . If this is the case, it can be said that T contains P and that establishes the notion of a match. More formally, it can be proved that there is a substring of T starting at some index l that matches P on a character by character basis, so that $T[l]=P[0]$, $T[l+1]=P[1]$, ..., $T[l+m-1]=P[m-1]$. It is outside the scope of this thesis to provide an exhaustive discussion of pattern matching algorithms. The reader is therefore urged to consult Goodrich and Tamassia's [24] practical overview of pattern matching algorithms for further details.

State transition analysis [25] is also a popular approach for representing and detecting known penetration scenarios. A penetration is modelled as a sequence of actions performed by an attacker indicating a clear path from the initial state to a target compromised state. An extension of this method that provides advanced correlation of intrusion signatures to infer intrusive activity is a '**coloured Petri-net**'[26]. This method represents intrusion states by using coloured tokens. The colour of the token in each state serves to model the context of an event. The signature matching is driven by the parsing of audit trails and is formed by moving tokens progressively from initial states to the final state that indicates a compromised system.

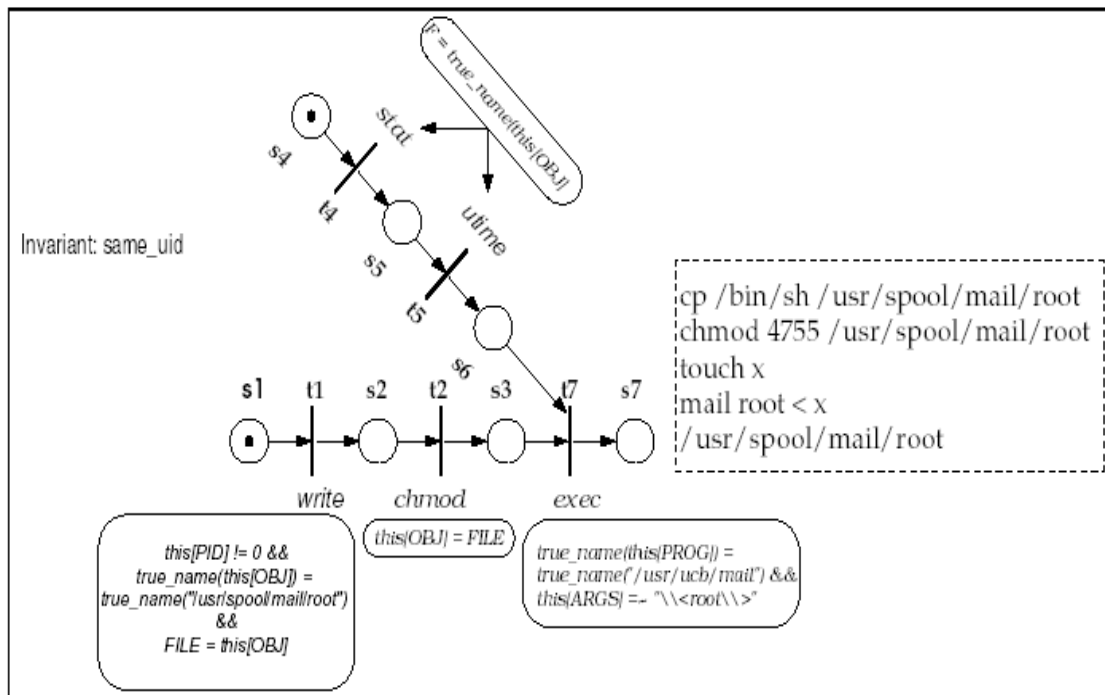


Figure 2.8: A colour Petri Net Automaton [26]

In order to further explain the application of Colour Petri Nets to misuse-detection orientated Intrusion Detection, figure 2.8 above illustrates a Colour Petri-net Automaton (CPA) that represents an intrusion signature. The system states (s1-s7) are represented with circles, whereas the directed arrows indicate the state transitions (t1-t7). In this particular example, an attacker is trying to invent a way of bypassing the authentication mechanism of a UNIX-based host and obtain System Administrator program execution privileges. The CPA-based attack signature dictates that the intruder should first insert a binary into the mail spool of the Super User (root) account. Then, the intruder will try to force the super user to execute it by means of checking his e-mail (s7). At this point, the system has been successfully compromised.

The primary advantage of CPA-based intrusion signatures is that they provide a very systematic way of defining detailed pre and post conditions for the matching of certain events that might indicate an intrusion. This creates not only a more refined-way of creating intrusion signatures, but it also introduces some variability on the attack signature, so that certain variations of an attack scenario can be encoded. This property addresses the inefficiency of detecting attack variants but it still cannot

address the greatest weakness of a misuse detection mechanism: its ability to detect a totally new method of intrusion [26].

2.5 Anomaly versus misuse detection and the birth of hybrid IDS frameworks

During the very early stages of IDS development, the community of researchers was always arguing about the optimum suitability of either anomaly or misuse detection for specific problem domains. Adopters of misuse detection were presenting their case by emphasising the computational effectiveness of pattern matching algorithms and its ability to offer fast detection results.

In contrast, the supporters of anomaly detection were focusing on the inability of misuse detection to detect attacks that have not been described in signature databases or intrusive activities that could bypass misuse detection by introducing minor differences in the execution of an attack. This is certainly one of the greatest disadvantages of misuse detection methods, since intrusion methodologies evolve all the time and produce new methods for attacking computer system infrastructures.

In a paper that describes several weaknesses of Network Intrusion Detection methods, Ptacek and Newsham of Secure Networks [27] describe in detail a method of camouflaging suspicious network traffic, in order to avoid detection from a network-based misuse detection engine. Figure 2.6 illustrated how information is encapsulated into the payload section of a Protocol Data Unit (PDU). Suppose that the string *“/etc/shadow”* indicating some sort of manipulation of a UNIX system password file was inserted into the payload area of the packet. All network-based IDS would intercept that string and flag an alarm, as a result of a rule that instructs them to match this particular string (or certain variations of it). However, the clever attacker inserts some characters into the string, so that each letter of the string is followed by a specific character. For instance, if that character was X, then the string would become: *“/XeXtXcXsXhXaXdXoXwX”* and would be an adequate measure to confuse the Network IDS engine.

From a philosophical point of view, neither of these methods is ideal for a range of scenarios. The ‘one-size-fits-all’ rule was never successful in the IT industry and that is certainly the case with Intrusion Detection Systems. An IDS should be a tool that addresses a plethora of different scenarios.

Whether the problem domain is related to detecting well-known attacks or suspicious behaviour, an IDS should be able to integrate a variety of different algorithms to address an ever increasing range of IT security issues. All previously referenced methods document a large number of failures under different conditions. These failures appear to be as **false positive alarms**, when the IDS flags a non-intrusive event as intrusive. The opposite (**false negative**) situation is equally undesirable, because a truly intrusive activity will be flagged as normal and consequently will go unnoticed.

As a result, IDS research and development started focusing on architectural frameworks rather than algorithmic investigations. An **IDS Framework (IDSF)** is essentially a holistic and abstract architectural specification. Amongst other things, this shift in IDSF research and development efforts introduced the effective combination of misuse and anomaly detection techniques for reducing the number of false positive/negative alarms and improving the reliability of Intrusion Detection Systems. They also focus on system-wide implementation issues, as the abstract properties allow the architecture to function with more than one IDS technique, Operating System or hardware platform, enhancing the interoperability of the architecture.

'**Haystack**' [28] was one of the earliest examples of IDSF frameworks. It was developed by Tracor Applied Sciences and Haystack Laboratories for the United States Air Force and employed a two part statistical anomaly detection procedure. The first part was sampling aspects of a user session and tried to determine the degree to which the session resembles an established intrusion type. The later stage was complementing the results of the first one by detecting deviations in a user's session activities from the normal user profile.

Denning and Lunt's work on the **Intrusion Detection Expert System (IDES)** [13] is another example of an architectural framework. Based to large extent on Denning's Intrusion Detection Model [8], IDES proposed a user behaviour classification model in terms of '*measures*', singled aspects of a user or subject's behaviour on the monitored system. These metrics were further classified into 'ordinal' or 'continuous', depending on whether they could be expressed in terms of a numeric count or quantification of the measure. IDES was the first statistical model that was independent of any particular system, application environment, system vulnerabilities or type of intrusions.

In January 1997, researchers funded by the Advanced Research Project Agency (ARPA) finished the specifications for the first **Common Intrusion Detection Framework (CIDF)** [29]. This was a major step towards establishing an architectural framework that focused on the issue of interoperability amongst different Intrusion Detection systems. A large part of the CIDF specification is dedicated to the process of establishing a standard way for describing intrusion events and directing IDS responses. In addition, an Applications Programming Interface (API) is defined as a reference for IDS software engineers, as well as a Specification Language to describe intrusive activity.

Despite the fact that the CIDF specification was designed to act as an interoperability tool for the IDS vendor community, at the time of writing, it has not been widely adopted in the commercial or academic world. Moreover, the development of the CIDF specification appears to be currently halted. While nobody can safely identify a reason for the fate of this interesting Computer Science experiment, certain aspects of the research effort have been taken over by a new group of the Internet Engineering Task Force (IETF). The Intrusion Detection Working Group (IDWG) [30] is a relatively new research effort focusing mainly on an IDS component message exchange framework, producing a variety of extensive specifications for IDS exchange and message implementation.

One of the latest IDSF efforts that we should also note is the Furnell and Dowland's **Intrusion Monitoring System (IMS)** architecture [31]. The architecture follows the principles of the previously mentioned IDSF research efforts, in that it combines both anomaly and misuse-based intrusion detection techniques and has a certain level of abstraction focusing on the way IDS components should be combined, in order to improve detection efficiency.

However, the IMS architecture has many novel features and it is worth emphasizing one that is the most important for this research project. In the early eighties, Anderson [33] has identified the need for handling not only intrusive activities originating from unauthorised users, but also events generated by legitimate users that abuse their privileges. The IMS architecture is the first IDSF effort that goes one step further by indicating a framework to handle this issue.

Apart from the academic and generic research and development IDS concepts, commercial vendors have produced their own design paradigms. Appendix A contains a generic overview of selected commercial IDS products available at the time of writing, with emphasis on outlining their generic design philosophy.

2.6 Threats: Definition, Detection and the concept of threat estimation

This thesis is concerned with predicting threats. Whilst earlier sections of this chapter have presented the concepts of computer security and computer intrusions, they have not explained what a threat really is and how it relates to the overall Intrusion Detection process. Pfleeger et al [32] defines the term threat in an IT infrastructure context as “a set of circumstances that has *the potential* to cause loss or harm”. These circumstances might involve human-initiated actions (intentional IT intrusions), flaws in the design of the computer system and environment factors (natural disasters).

However, as the aforementioned definition states, threats do not always evolve into harmful situations. A threat’s potential is realized by the exploitation of a number of weaknesses in the design of the IT infrastructure (software, hardware, management procedures, location). These weaknesses are called *vulnerabilities* [32]. One can then distinguish the relationship between threats and vulnerabilities: A threat turns into a harmful situation by means of exploiting one or more vulnerabilities.

In order to illustrate the difference between these two concepts, it is useful to consider an example in Data Security context. The fact that a potential cracker is skilled and desires to break into an organization’s IT infrastructure is a threat that will not always materialise into a successful intrusion. On the other hand, if your company’s systems are lacking updated software, monitoring software and/or the care of a professional system administrator, a window of opportunity is created for the cracker by these vulnerabilities.

Most of the IDS designs address the problem of tackling intrusions of external origin. The following two Chapters of the thesis will elaborate more on the anatomy of internal intrusions, which is the thematic area of this research project. Appendix A outlines a selection of IDS products that specialise in detecting the problem of internal intrusions. This is a positive step towards the handling and isolation

of insider cases. However, the mere detection of an internal intrusion is not a panacea in the process of managing these kinds of threats. A way to predict these kinds of threats would also be a valuable asset of an Intrusion Detection System.

The process of predicting a particular set of events in order to prevent their occurrence and provide a better understanding of their underlying mechanisms does not represent a new methodology in the field of science. Many scientific disciplines have introduced prediction mechanisms that have a number of applications. The utilisation of game theory in financial forecasting [47] in order to predict the value of shares in the stock exchange market and the processing of seismic data for oil discovery purposes [48] are notable examples of models that already serve our world and used on a daily basis by analysts, as value-added tools that help their research.

In the same way, a process that provides an estimate of emerging internal threats by modelling certain factors would be a useful tool for a Data Security analyst. The International analyst firm Gartner estimates that by the year 2005, 60% of security costs of a business enterprise environment will be due to insider attacks [49]. An effective Insider Threat Prediction methodology would help data security specialists identify individual factors that are likely to produce these threats. It could be a value-added component of an existing Intrusion Detection System, instructing it to increase the intensity of monitoring only for specific machines or users and hence increasing its efficiency. At the time of writing, no known methodology exists in order to establish a suitable Insider Threat Prediction Model.

Thus, the epicentre of this research project is the task of deriving a suitable model that utilises threat detection techniques, in order to facilitate the prognosis of insider IT misuse occurrence. The research considers both insider threats (motive, skill and other factors) and vulnerabilities (mechanisms that the insider IT misuser exploits in order to successfully breach a system). As a result, the context of Data Security for this research project is the process by which we provide proactive capabilities to help the system prognose insider threats, in order to safeguard its proper operation.

2.8 Conclusions

This chapter provided an overview of the birth of the Intrusion Detection System, the concept of Computer Security, threats and vulnerabilities, as well as a discussion of the major IDS techniques. Appendix A offered an overview of commercial IDS paradigms. After the presentation of these concepts, three issues should be clear at this point:

- None of the major IDS techniques (anomaly and misuse detection) represents a panacea for providing an efficient IDS system that would result in a minimum number of false positive/negative alarms.
- The shift of focus from developing pure IDS techniques to IDS Frameworks is still under intensive development, with IDS vendor interoperability and Intrusion Specification issues not being substantially addressed.
- The development of research frameworks that will specifically address the issue of managing insider threat by predicting legitimate user intrusive activities has largely not been addressed by the IDS community at the time of writing.

The last point forms the main argument for the motivation of this research project. Consequently, the next logical step is to start analysing the legitimate user problem in more detail. The next Chapter of the thesis introduces the reader to the concept of the legitimate user misuse problem. Essential terminology is introduced as well as references to relevant cases and surveys, in order to provide an estimate of the magnitude of the problem.

CHAPTER 3

COMPUTER INTRUSIONS AND THE ‘INSIDER’ IT MISUSE PROBLEM

Misuse: *to use (something) in a wrong way or for a wrong purpose*

Longman Dictionary of Contemporary English

Previous chapters considered the concepts of computer intrusions, threats and vulnerabilities. It is now time to examine their manifestation in the real world. After familiarising the reader with essential terminology, this chapter will present various statistics that provide information about the frequency of occurrence and type of computer intrusions. The figures were taken from recent and highly regarded information security surveys. A subset of computer intrusions is related to IT misuse incidents that originate from legitimate users. Some surveys simply mention these types of incidents, whereas others consider them to a greater extent. However, the main goal of this chapter is to prove that the importance of the insider IT misuse problem has been undervalued by critically evaluating the statistical figures and examining the real amount of information they reveal.

3.1 Towards qualification of insider IT misuse acts

The ‘insider IT misuse’ problem has two main thematic areas. One of them relates to the term ‘insider’. At the time of writing there is no consistent definition throughout the information security literature for it. The earliest attempt to classify internal misuse of computer systems is presented by Anderson [33] and discusses borders of distinction amongst ‘**masqueraders**’, ‘**misfeasors**’ and ‘**clandestine**’ users.

‘Masqueraders’ are insiders that exploit weaknesses of the authentication modules of a particular application or Operating System, thus gaining the identity of other legitimate users. A ‘misfeisor’ is an insider that does not need to masquerade, but abuses the power of his/her privileges to alter maliciously the operation of the system. A ‘clandestine’ user is related with authorised users and their capabilities to bypass audit, control and access resource mechanisms in a particular computer system. It is important to emphasize that the categories of masqueraders and clandestine users are really disguising as legitimate (i.e. authorised) users. Hence, although they are intruders that appear to act as internal elements of an IT infrastructure, they cannot be considered as ‘insiders’ due to the fact that they do not represent the people who are meant (authorised) to use the systems (misfeasors).

Some studies [50] tend to further classify insiders as logical and physical ones. A logical insider operates physically outside the context of an organisation. For instance, consider the case of an employee that uses telnet to connect to his UK company transaction server from China. Other factors, such as operating system authentication techniques, as well as the environment of the user might differentiate amongst logical insiders. On the other hand, a physical insider would connect to the same server, within the physical bounds of the IT infrastructure of the organisation (including buildings, or external trusted networks referred to as extranets). However, if we consider the increased levels of connectivity offered by the convergence of mobile computing and telecommunications platforms, the previously mentioned classification scheme will become less apparent in the near future.

The distinction between an insider and an outsider can be vague when it comes to authentication mechanisms. Assuming traditional password-based authentication mechanisms that, at the time of writing, constitute the norm for authenticating users on most Operating Systems, and the fact that a successful outsider might be able to bypass them successfully by exploiting vulnerabilities, users that give away their passwords and other techniques, there comes a point when an outsider becomes an insider. From a system point of view and depending on the skill of the external attacker to emulate the behaviour of a legitimate user, there might be no difference between an outsider and an insider.

Instead of conforming to the previous ambiguous interpretations of the term 'insider', a more suitable conventional interpretation is proposed. An insider is a person that has been legitimately given **the capability of accessing one or many components of the IT infrastructure**, by interacting with one or more authentication mechanisms (plain text password, PKI, biometric or smart card token). The word 'legitimately' has been underlined because it emphasises the main difference between an insider and an external cracker. An insider should always be able to have at least a point of entry in one or more computer systems. The implications of having such a point of entry is that an insider does not usually need to consume as much time and effort to obtain additional privileges as an external cracker does, in order to exploit IT infrastructure vulnerabilities and mount an attack. It also means that an insider is less likely to get caught by implemented security measures because of the level of trust that he/she enjoys. These aspects make the problem of tackling insider IT misuse a composite and difficult one. Latter paragraphs will illustrate this fact with appropriate case studies.

Pfleeger et al [32] mentions the security acronym 'MOM' which stands for 'Method Opportunity Motive', indicating that there are many elements in an IT security attack recipe. The 'Method' term signifies the skills, knowledge, tools necessary to complete an intrusive activity. 'Motive' is the actual reason to perform the attack (trade secret theft, forcing a company to lose revenue, revenge are examples of potential motives). Finally, the term 'Opportunity' relates to the time and access to accomplish the attack. An outsider and insider might have similar motives and skills, however their respective opportunity chances to mount an attack are different. As the previous paragraph explained, an insider needs less effort and enjoys a greater level of trust than an outsider.

The other side of the 'insider IT misuse' problem relates to what can be considered as misuse activity. Although the great majority of the people are familiar with the generic meaning of the word 'misuse', when we try to map it to an insider IT context, there is a need to clarify certain issues. Insider IT misuse can be a very subjective term. In fact, one of the most challenging tasks is to draw a clear line that separates an IT misuser from a person that uses the available resources in an acceptable way and for an approved purpose. The words 'acceptable' and 'approved' imply the presence of rules that qualify (or quantify) conditions of allowable usage for the resources concerned. These rules are often embodied within an IT usage policy. Part of this organisation-wide policy is the information security policy, defined as the '*set of laws, rules, practices, norms and fashions that regulate how an organisation manages, protects, and distributes the sensitive information and that regulates how an organisation protects system services*' [51].

Different organisations pose different restrictions on IT usage, and this variety of rules adds a considerable level of ambiguity to the term 'misuser'. In order to overcome this uncertainty, it is necessary to introduce a taxonomy of insider misuse incidents. The derivation of such a taxonomy will systematise the deployment of an information security policy across an organisation and is a necessary step for advancing the research on insider IT misuse. However, we shall not discuss a suitable taxonomy here. Instead, chapter 5 of this thesis presents a suitable insider misuse classification scheme.

3.2 Towards quantification of insider IT misuse acts

The quantification of the magnitude of the insider IT misuse problem is a difficult process. One has to start by looking at general computer intrusion figures that are widely available and then try to isolate data that are relevant to activities initiated by insiders.

The British Department of Trade and Industry (DTI) in association with PriceWaterhouseCoopers (PWC) published the 'information security breaches survey 2004' [52]. The survey mentions that Insider Misuse has doubled since the year 2002, mainly driven by the increased adoption of World Wide Web and Internet related technologies. Approximately a third of the DTI/PWC 2004 respondents claimed that their worst security incident was internal. This is clearly another verification of the existence of internal security threats.

Figure 3.1 displays the distribution between internal and external incidents in the DTI/PWC 2004 survey for small, medium and large organisations. Whilst the smaller IT infrastructures appear to face more incidents of external origin, the gap between insider and outsider incidents is smaller for respondents of medium and large scale organisations. This indicates that the likelihood of IT misuse from legitimate users is a very probable scenario.

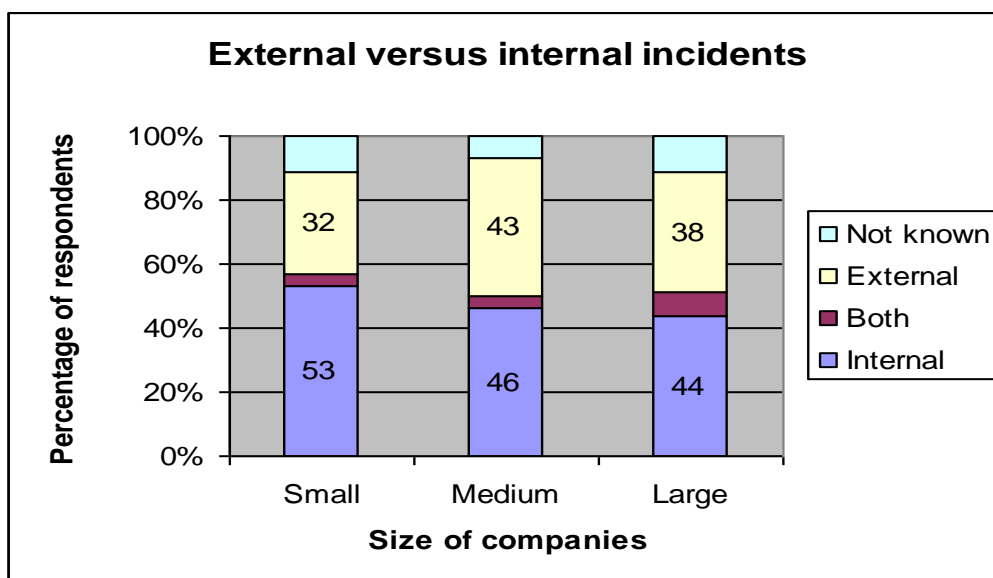


Figure 3.1: External versus internal incidents in terms of report frequency [52]

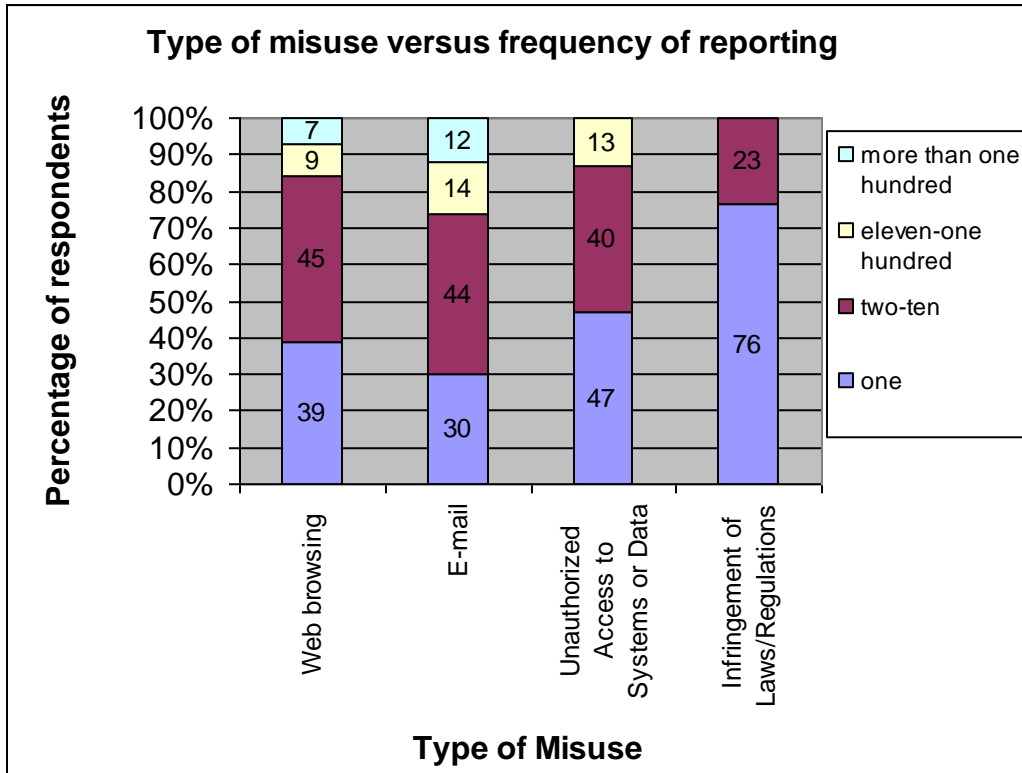


Figure 3.2: Types of misuse reported by UK businesses [52]

Figure 3.2 displays the type of legitimate user misuse reported by UK businesses [52]. The misuse of World Wide Web and email facilities are the most frequent type of insider misuse activities. Excessive usage of these facilities for personal use as well as for viewing and disseminating inappropriate material were considered by the DTI/PWC survey as misuse incidents for web and email facilities.

The ‘Computer Crime and Security Survey’ of the San Francisco-based Computer Security Institute (CSI) [53] is another survey that also emphasizes the presence of insider threats. The survey makes clear that for the last seven years of its research scope, computer intrusions have formed a substantial threat for IT infrastructures. In the year 2003, ninety percent of respondents detected computer security breaches within the last twelve months. More than three quarters (78%) of the participants cited their Internet connection as a frequent point of attack. It should be also noted that the rising frequency of computer intrusions is also accompanied by substantial financial losses associated with them. Approximately forty seven per cent of the 2003 survey’s respondents were willing to quantify their losses to a total sum of 201,797,340 US dollars. This amount is 55% lower than the year 2002 estimated total annual losses. However, if someone takes into account the fact that the 2003 annual loss figure does not include the remaining fifty-three percent of the participants that were not willing

(or able) to estimate their losses, it is reasonable to assume that the real cost of computer intrusions is considerably higher than the reported one. This is true not only for the year 2003 but also for the estimated annual loss figures of previous years.

The next big question to answer is what poses a greater danger to an IT infrastructure: Are insider misuse incidents more dangerous than the ones caused by the acts of external hackers? The 2003 CSI/FBI survey contains useful figures that are analysed in the following paragraphs.

The 2003 CSI/FBI survey figures are not accompanied by any commentary on the issue. This was not the case for earlier editions of the same survey. The 2002 CSI/FBI Computer Crime [54] survey debates the issue. The director of CSI Patrice Rapalus states that the survey “*has challenged some of the profession’s ‘conventional wisdom’, for example that the ‘threat from inside the organisation is far greater than the threat from outside the organisation’...*”, based on the fact that the overall number of the reported insider incidents has dropped.

The shift of perceived threat from insiders to external hackers was also noted by Dr. Dorothy Denning. In the 2001 CSI Computer Crime and Security survey [55], she wonders about the dropping frequency of insider incidents by stating: “*For the first time, more respondents said that the independent hackers were more likely to be the source of an attack than disgruntled or dishonest insiders (81% vs 76%). Perhaps the notion that insiders account for 80% of incidents no longer bears any truth whatsoever.*”

On the other hand, Dr. Eugene Schultz [55] has a different opinion about the way the CSI report presents the importance of the insider threat, clearly challenging the CSI survey: “*Is it that we should ignore the insider threat in favour of the outsider threat? On the contrary. The insider threat remains the greatest single source of risk to organisations...*”

This diversity of opinions represents one fundamental question about the insider misuse problem. Should someone weight its importance in terms of its occurrence frequency or in terms of the potential consequences that this particular type of incident might have? The truth lies in the statistics presented in the CSI survey. If someone analyses them carefully, some interesting patterns will be revealed.

The graphs of Figure 3.3 report external and internal (insider) incidents for the last four years (2000-2003). The thing to note is that there is a small difference between the number of respondents that reported an external incident and those who reported insider events. This was also reflected by the figures of the DTI/PWC survey for medium and large scale organisations.

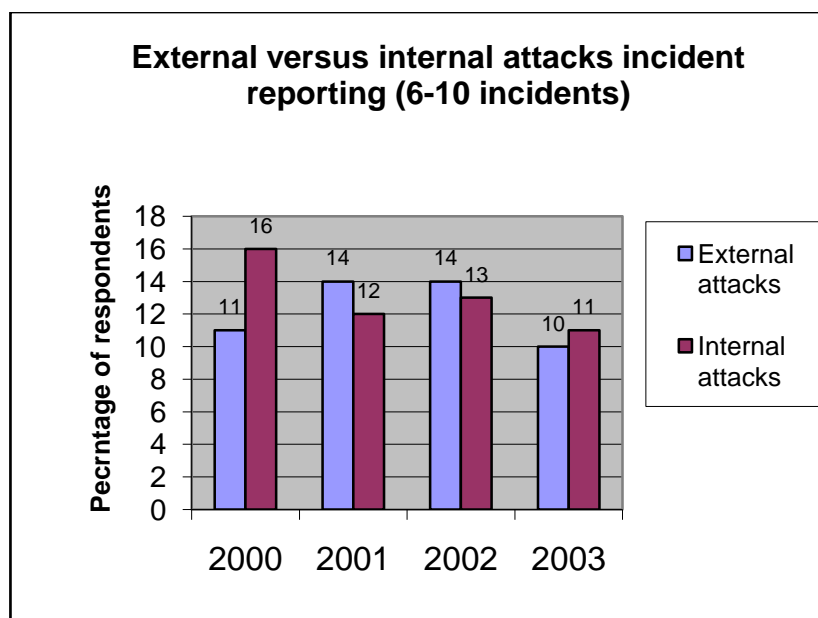
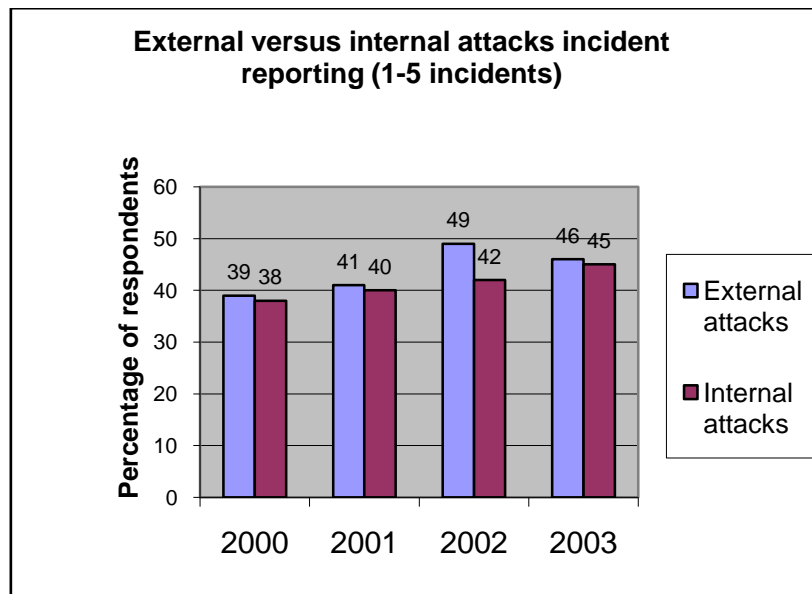


Figure 3.3: External versus internal attack incident frequency (source [53])

The CSI/FBI 2003 survey provided an estimated cost for various types of computer crimes. Appendix B of the thesis contains the survey table ‘The cost of Computer Crime’, where one can view aggregate costs sampled over a 48 month period (2000-2003). Whilst the table is a useful generic indicator of the

financial impact of computer security incidents, it cannot be used to provide a safe comparison between insider and external incident costs. This is because very few of the incident categories mentioned in the table can be attributed exclusively to legitimate or external perpetrators.

For instance, categories such as “System Penetration by Outsider”, “Insider abuse of Net access” and “Unauthorised insider access” can be safely used to relate the cost of security incidents to external or internal origins. In contrast, the rest of the incident categories could be attributed to both internal and external origins. This fact combined with the small percentage of the survey respondents that were able to quantify their losses (just 47% for 2003) makes the comparison between internal and external incidents unfeasible.

Commercial security software vendors have also started warning about the emerging problem of insider threat. Although someone suspects that these surveys might have a strong sales-orientated bias, they still represent a good marginal picture of the problem. TecSec [56] is a company specialising on preventing confidentiality breaches that result from legitimate user actions. They use a method known as ‘Constructive Key Management(CKM)’ to control file access to and from a particular group of users. The vendor quotes that approximately 70% of data loss originates from accidental or planned breaches of security policies by employees.

‘Rapid7’ [57], a New York based security software vendor that specialises in system penetration testing has also summarised its results on computer intrusive incidents. In their independent 2001 ‘Network Security Survey’, the vendor has queried more than 160 US based companies and government agencies. Rapid 7 estimates that during the year 2000, US businesses lost approximately 1.6 trillion US dollars. The survey does not justify the exact source or the way the vendor derived this amount. The figure seems exceedingly large when compared to the statistics quoted by similarly-minded surveys (the CSI estimates a total loss of approximately 1.5 billion US dollars for the period 1997-2001). However, it does state clearly that a marginally higher percentage of its participants believes that they are more at risk internally (31%) than externally (25%), whereas the highest percentage (41%) believes that they are equally at risk from both internal and external factors. The remaining 3% did not have an opinion about the most probable origin of threats.

3.3 Trust, insider IT misuse and some notable real-world cases

Apart from the alarming figures quoted in several surveys, someone can grasp the real threat potential of insider IT misuse by considering one fundamental aspect of every insider: The level of trust he/she enjoys in a particular organisation. It is this level of trust that makes the detection of legitimate user misuse difficult to detect and deal with. The best way to illustrate this difficulty is by briefly mentioning a sample of real-world insider cases that have received widespread attention.

The 2001 CSI/FBI survey [55] cited the case of Robert Hanssen, a 56 year-old FBI veteran. Hanssen abused his trusted access to the FBI Automated Case Support System that contained classified information about ongoing investigations and handed critical information to Russian agencies. In return, he was receiving large sums of money, inflicting a great deal of damage upon the prestigious image of the Federal Bureau of Investigation and the national security of his country. Nobody could imagine that a church-going and patriotic family man was betraying his country for money.

Hanssen's seniority and level of trust were not the only weapons that helped him to remain unnoticed. Having developed a more than average level of IT knowledge, he utilised an unusual way of hiding the information he wanted to trade with Russian agents. Hanssen was using specially formatted 40-track mode diskettes, in order to hide the sensitive information in (what appeared to be) a blank area of the disk. Even if someone wanted to inspect the floppies he was using for his personal data backup purposes, it would have been difficult to discover the hidden information without the usage of an advanced data forensic tool. The combination of his colleagues' trust and his own data hiding techniques allowed him to operate for certain number of years inside various FBI facilities.

The case of Abdelkader Smires [58], a chief software engineer who worked with Internet Trading Technologies is a typical example of what can be achieved by a disgruntled insider. Smires had financial differences with his employer. He thought that he was underpaid and requested a pay rise coupled with a range of additional benefits. When his requests were turned down, he decided to take revenge by using the computers of his previous employer (Queens College) to launch a Denial of Service (DoS) attack. His actions caused several hours of downtime (and lost revenue) over a three-day period for his employer.

There are two important points to consider with regards to the Smires case. The first fact is that he had legitimate access to another organisation (Queens College) due to an account that should have been erased a long time ago. This allowed him to conceal (at a first stage) his attack on Internet Trading Technologies. The second and most important point is the level of knowledge he possessed about Internet Trading Technologies' IT infrastructure. Smires knew which IT components are likely to be vulnerable to certain methods of DoS attack. Hence, it was very easy for him to be able to disrupt the functioning of the computer systems.

Garfinkel and Spafford [7] mention the 'Leeson-Iguchi' case. Nick Leeson ('Barings' Bank – Singapore) and Toshihide Iguchi ('Daiwa Bank' - New York) were investment traders working together for two major financial organisations. They made risky investments and lost large amounts of investment capital. However, instead of admitting their losses, they illegitimately modified computer records to cover their mistakes and continue to be able to request vast amounts of money to invest. As a result, Barings Bank was forced into insolvency and 'Daiwa' lost its entire United States customer base. More than 1 billion dollars of investment capital vanished as a result of their actions.

Barings Bank had an internal data audit mechanism that focused on discovering potential external breaches, without focusing on insider actions. Clearly, they have underestimated the insider threat factor. They could never think that two accountants that had direct access to database records of investment funds would commit fraud in this way. This electronic record forgery would probably go unnoticed if Leeson and Iguchi managed to stop their losses. They did not and consequently the large sums of unaccounted investment capital forced an internal investigation that revealed their actions.

E-mail abuse is a different part of the insider misuse spectrum and is revealed by the Norwich Union versus Western Provident Association case [59]. A Norwich Union employee circulated an e-mail that contained what could be considered as a sarcastic (or defamatory) rumour about Western Provident going into financial difficulties. The e-mail leaked outside the company (another internal user thought it was a great joke) and eventually came to the attention of the rival company. Consequently, Western Provident took legal action against Norwich Union and the case was settled with the latter paying

approximately £450,000 pounds in compensation plus the legal expenses for Western Provident. What was initially considered as an innocent joke proved to be the reason for commencing a rather expensive legal case.

3.4 The borders between internal and external cases

After qualifying and (attempting to) quantify the magnitude of the insider misuse problem, one might attempt to draw a limit between what can be classified as an external incident and what can be considered an internal case. However, one real-world case might offer an alternative view that defies this dualistic approach.

In the morning of the fifteenth of November 2002, the Computer Emergency Response Team (CERT) at the University of Oslo Computing Services decided to reset the passwords of approximately 52,000 campus users [60]. The decision was taken after the disturbing discovery that a group of German hackers have managed to gain access to several user accounts, ‘masquerading’ as legitimate system users. After the cumbersome process of issuing tens of thousands of new user passwords, extensive forensic examination of several servers took place, trying to establish the method that won the hackers access to the service. Despite the employment of various information security mechanisms, the weak point was traced back on a third party telephony database product. An external contractor was testing the telephony database, but when the system became operational, the contractor forgot to change the trivial administrative password of the system.

The previous scenario represents no unusual elements (similar administrative mistakes are likely to occur often). The press and the University management authorities have treated the case as an external hacking incident. On the other hand, someone could argue that there were several internal factors that have contributed substantially towards the establishment of the breach.

The question on how someone should classify security incidents is a rather philosophical one. If someone wishes to use the point of origin as a classification criterion, then it would be right to characterize the University of Oslo incident as an external one. However, the point of origin is not necessarily the best criterion for understanding why security incidents emerge. The enabling

mechanisms that allow an external entity to penetrate the defences of an IT infrastructure are also important and they might be related to internal factors. In this case, the external contractor as well as the responsible local system administrator who did not supervise a machine that was connected to an operational network, were at the time a liability for the organisation in question. This could happen due to excessive workload, lack of training, bad communication or lack of appropriate regulations. The main point is that insiders that accidentally (or even deliberately) do not perform their job properly can constitute a substantial threat for the IT infrastructure of the network.

Hence, it is fair to say that for every external case there might be a range of internal factors that contribute to the establishment of a successful information security breach. **Mutual exclusion is not always applicable in this domain** and a comprehensive Insider Threat Estimation process should take these factors into account.

3.5 Conclusions

All the previously mentioned cases represent common trends of the insider IT misuse problem. The list is by no means an exhaustive one. Chapter 5 will elaborate and classify more systematically the different types of insider IT misuse. Here we only provide a representative sample. All cases of this sample proved to be very expensive mistakes for the organisations associated with them and they all had one thing in common: The insiders that committed the misuse activities were all blindly trusted and they were acting beyond suspicion.

Hanssen was often described by his colleagues as a “Church going, family man”. Nobody (including the highly trained FBI officers) thought that this man was selling national secrets for money. Smires was trusted (despite the fact he had no substantial reason to have an IT account) by the system administrators of Queens College because he used to be an academic member of staff and he knew the vulnerable points. Leeson and Iguchi were highly respected traders, and hence nobody bothered to check their computer account records for inconsistencies. The Norwich Union case shows that insider threats can be accidental, showing that large corporations ignore the power of communicating to the external world via an IT infrastructure. Finally, the contractors and system administrators at the University of Oslo case were expected to contain these vulnerabilities effectively and the case logically

proves that some cases that have been characterised as external breaches contain some sort of internal misuse element.

Alarming figures that indicate only a fraction of the real magnitude of the problem and high-profile real-world cases constitute the picture of the growing insider threat. An important conclusion that can be derived from the previous discussions is that the frequency of occurrence of a particular type of incident should **not** be used as the only measure of the level of threat it constitutes for a particular organisation. Indeed, the figures might disprove conventional wisdoms of the type ‘80% of security incidents come from insiders’ but they really undervalue the importance of the insider threat.

In addition, it was shown that for what people classify as external attacks, there is always a range of internal factors that open the way for hackers. This last consideration prompts for a radical change in the philosophy we classify internal and external security incidents.

For all these reasons, a more systematic examination of the Insider Misuse problem is needed and the best way to achieve this goal is to produce a survey that targets specifically insider misuse. The next chapter of this thesis discusses the scope of devising such a survey and presents its results.

CHAPTER 4

THE INSIDER IT MISUSE SURVEY

After the discussion of more generic information security surveys, it is now time to target the problem of detecting Insider IT misuse by gathering more specific information about it and the technologies that attempt to address these issues. This chapter will present the thinking behind the design of the ‘Insider IT Misuse Survey’, as well as the collected results, in order to answer several fundamental questions:

- How popular are Intrusion Detection System technologies amongst IT professionals? Do they contribute towards the containment and prevention of computer intrusions?
- Are legitimate user incidents more frequent than external hacking attacks? Is the first type of incident more serious than the latter one as expected after examining earlier evidence in Chapter 3, in order to verify the necessity of researching the field of Insider IT Misuse?
- What really constitutes an insider IT misuse problem? What are the most frequent ways for a legitimate user to abuse an IT infrastructure? The answer to this question will greatly help towards forming ways to classify legitimate user misuse activities.
- What are the most likely places in computer systems to collect information about legitimate user misuse, in order to devise appropriate metrics for gauging the potential for IT misuse?
- Is there any indicative information about what kind of user is likely to initiate an insider IT misuse incident? Forming user profiles for specific misuse incidents can aid the construction of threat estimation techniques to a great extent.

Two methods were considered for delivering the survey to the respondents: traditional post and the World Wide Web. It was decided that the best way to deliver the survey’s questionnaire to the respondents was via the World Wide Web interface for several reasons. The time scales for devising, distributing, collecting and analysing the data were limited to 9-12 months. The entire research project’s lifespan was 36 months and time was needed for researching other aspects of the insider misuse problem. As a result, publishing the survey on a web page was the fastest (and most economical) way to collect a large number of results in the shortest possible time. In addition, it was easier for most of the respondents to complete the survey on-line and press the ‘Submit’ button to send their results, rather than filling a form and then go through the time-consuming process of posting it.

However, delivering a survey via the World Wide Web has its disadvantages when it comes to result accuracy, verifying the identity of the respondent and providing assurance about the protection of the respondents' submitted data. Several measures were deployed in order to address issues ranging from anti IP spoofing techniques to maximum database server security. Each participant's e-mail address was verified by sending automatically an e-mail, as a receipt of participating in the survey. E-mail addresses that were invalid were marking the participant as unreliable and the entire record was discarded. All valid records were then submitted to a Relational Database Management System (RDBMS) for further processing. The RDBMS engine was running on a different computer than the Web Server offering the survey's web forms to reduce the possibility of a data security breach.

The 'Insider IT misuse' survey ran for approximately nine months (August 2001 – April 2002) and targeted various IT professionals (system administrators, IT security specialists, technical managers /CEOs) across Europe.

Appendix C contains a copy of the survey's questionnaire, which consisted of 18 questions divided in three parts. Part A gathered generic information about the participant and his/her organisation, as well as information related to the usage of intrusion detection systems and firewall technologies. Part B aimed to compare the level of criticality of the insider threat to that of external hacking activities, in terms of frequency of occurrence, resulting financial damage and legal consequences. Finally, Part C collected more information about the nature of the insider IT misuse, providing useful insights on where and how the problem occurs more frequently.

In general, the number of questions was kept to a minimum to avoid inconveniencing prospective participants whose time was potentially limited. In order to persuade people to avoid being anonymous and hence have an additional way of validating the authenticity of the submitted results, a subscription service was offered: If the participants submitted a valid e-mail address, they would automatically be mailed back the results of the survey.

4.1 Who were the respondents

In overall, the survey collected data from 50 respondents of European origin. Although a greater number of participants were originally expected, this number still represents a valid sample as the survey displays clearly certain trends that could be used to analyse further the insider problem and make a profile of the average insider.

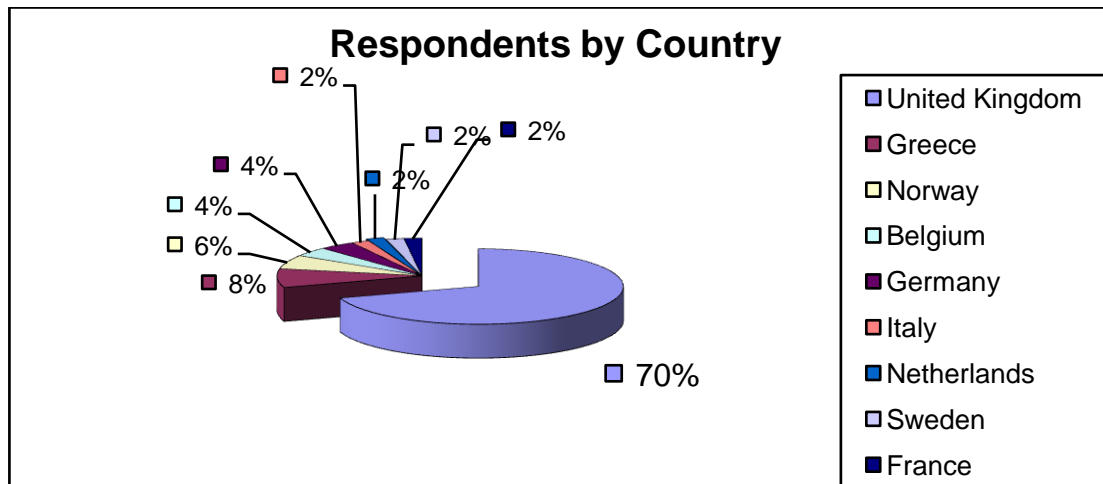


Figure 4.1: Participants' country of origin

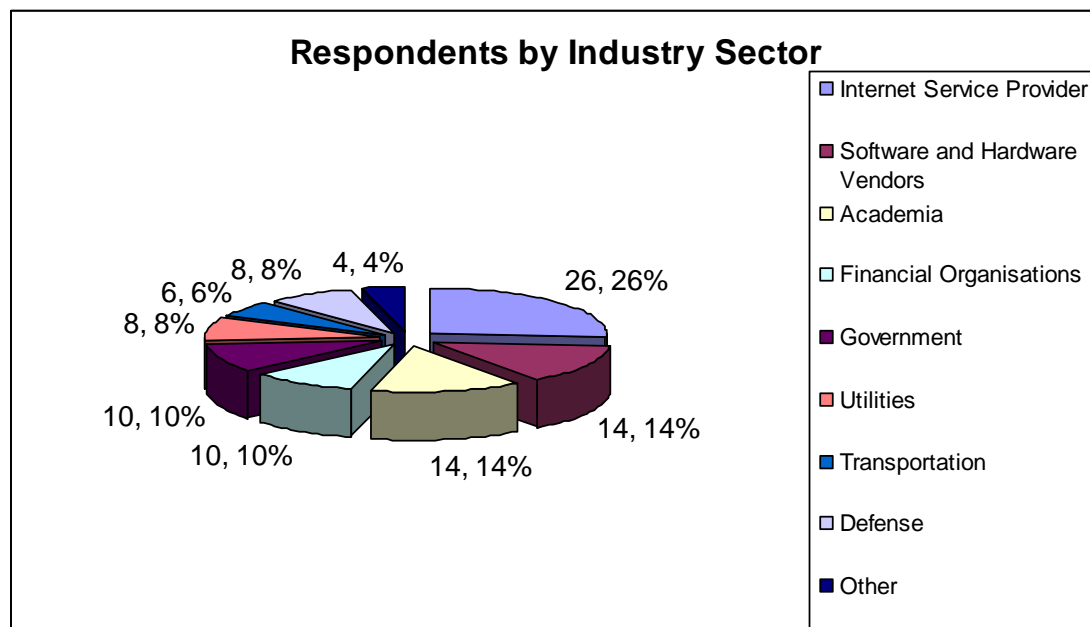


Figure 4.2: Participation sorted by Industry Sector

The bulk of the participants came from the United Kingdom (70%) as the graph of figure 4.1 indicates. The next graph illustrates the breakdown of respondents by industry sector, in response to question 3 of the survey: 'In which of the following IT sectors does your company/organisation belong?' Software

and hardware vendors came on top of the list (26%), whereas a smaller number of participants originated from 'utilities' and defence companies.

Question 1 of the survey ('What is your role inside your organisation/company?') probed the professional background of the participants (Figure 4.3). The great majority of the questioned IT professionals had technical background (system administrators-46%, system developers-24% and IT security consultants-18%), leaving a margin for non-technical opinions that originate from management (Human Resources, Executive Boards).

This does not necessarily provide a balance of opinions amongst technical and non-technical respondents. The thesis is after all concerned at large with addressing the problem of insider IT misuse at a technical level. Nevertheless, in an IT infrastructure there is always a direct relationship between what is enforced by technical personnel and the desired information security policy derived by management. Thus, an opinion from management would still be of great value for the respondent sample, especially for information security policy issues.

The majority of the participants came from medium to large-scale organisations (between 100 and 500 computer systems), as the final chart of figure 4.4 indicates. Finally, 47 out of the 50 respondents had more than 5 years of experience in their current role.

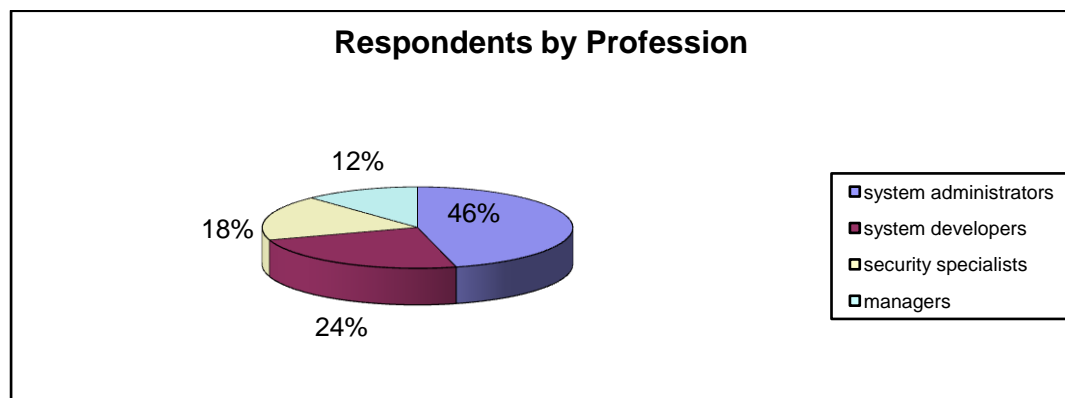


Figure 4.3: The professional background of the respondents

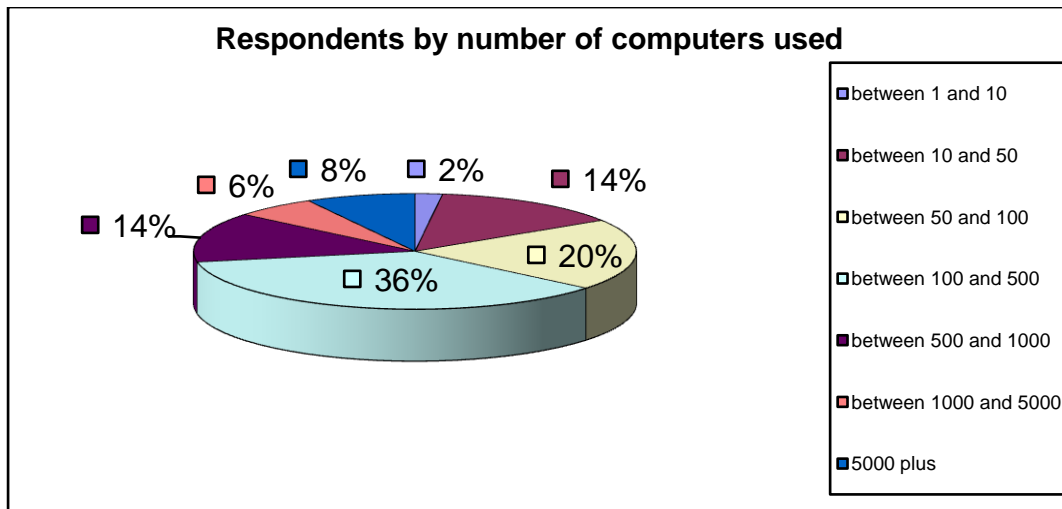


Figure 4.4: The respondents' size of IT infrastructure

4.2 Usage of security related technologies and reported incidents

Questions 5 and 6 of the survey tried to briefly estimate the deployment popularity as well as the perceived level of satisfaction from the operation of several Data Security technologies amongst the respondents. The participants were offered four possible answers for each of these two questions:

- “Yes, we use these technologies”: It is known that a particular range of technologies is employed, without the participant indicating major problems with their operation.
- “Yes, but they are not very effective”: It is known that a particular range of technologies is employed and it is also known that the respondent is not happy about their operational effectiveness.
- “No, but we are thinking of installing them”: This answer indicates that the participant has not deployed the technology, but she clearly thinks the technology is useful.
- “No, and we believe we do not need them”: The respondent indicates that the technology is really an unpopular choice.

The design of these questions would have been complete if the survey asked the participants to also justify the reasons for being dissatisfied with respect to the technologies in question. However, it was felt that this could substantially increase the amount of time it takes to complete the survey and could potentially act as a deterrent for the respondents. The main target was to keep the survey short and simple and focus on aspects specific to the Insider Misuse problem instead.

Chapter 2 of the thesis elaborated on the origins of Intrusion Detection Systems and proved that they represent a relatively new technological trend of the Information Security domain. This fact is reflected in the results of this survey. In response to the fifth question ‘Does your organisation employ a combination of 'firewall' and/or antivirus and/or data encryption product?’, nearly all (96%) of our respondents answered that this was the case (including those that were happy and unhappy with their operational effectiveness). 4% of the participants did not utilise any of the previously mentioned ‘traditional’ Data Security technologies. An analytical breakdown is illustrated in Figure 4.5.

On the other hand, the employment of Intrusion Detection Systems was more limited (Figure 4.6). The following question asked the respondents to comment on whether they have deployed an IDS solution. Only 22 out of the 50 (44%) respondents used Intrusion Detection. IDS deployments were always combined with traditional information security technologies (i.e. none of our respondents used exclusively IDS systems). This was somehow expected, given the fact that the corporate world is always slow to adopt new technologies in mission-critical production environments.

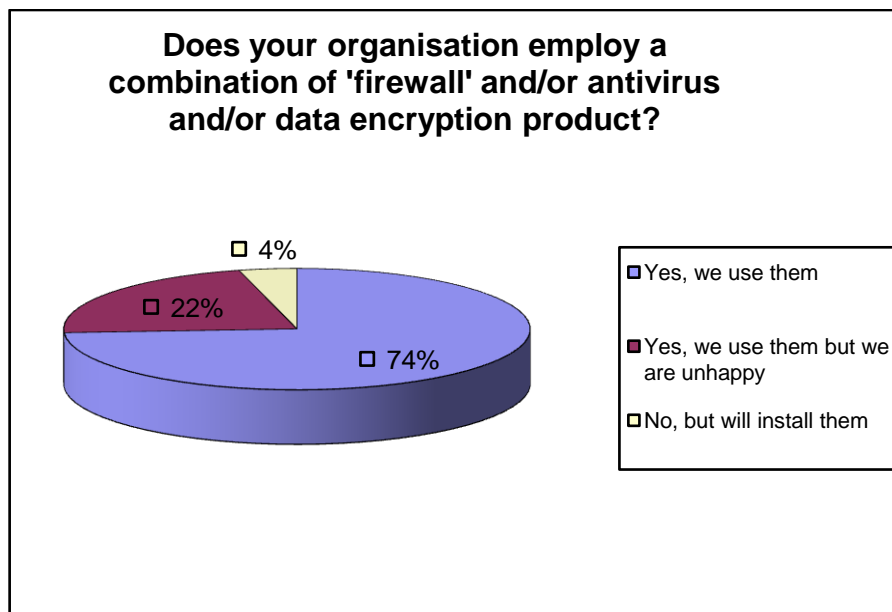


Figure 4.5: Popularity of traditional Data Security technologies amongst the respondents

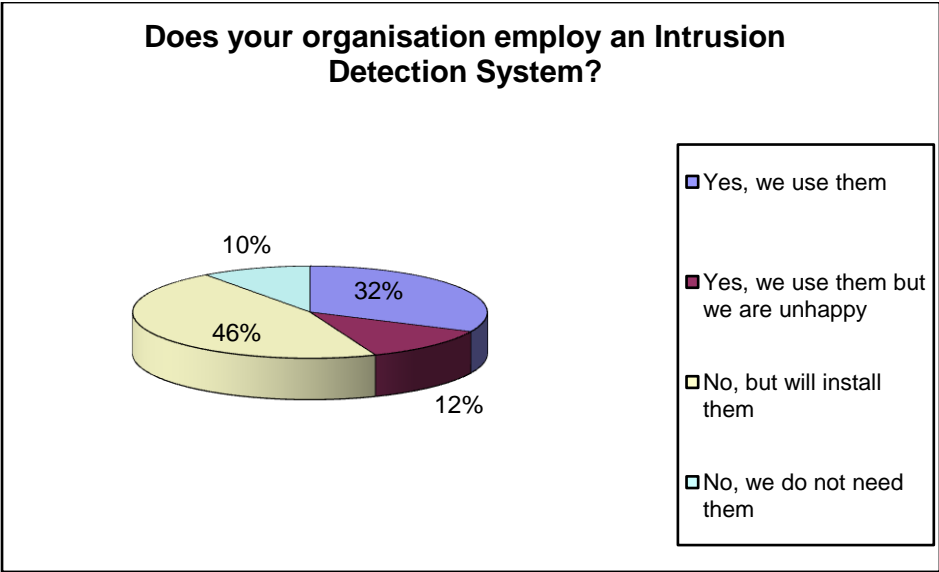


Figure 4.6: Popularity of IDS deployments amongst the respondents

With regards to the perceived level of satisfaction, it is evident that most users of traditional Data Security Technologies were happy with their deployment. In particular, 37 out of the 48 (77%) respondents that used these technologies believed that their operation was smooth with no negative impact on their business. Similarly, the perceived level of satisfaction related to the usage of Intrusion Detection Technologies was more or less on the same levels. 16 out of the 22 (72%) respondents that employed IDS solutions in their IT infrastructure reported no major problems with their operation.

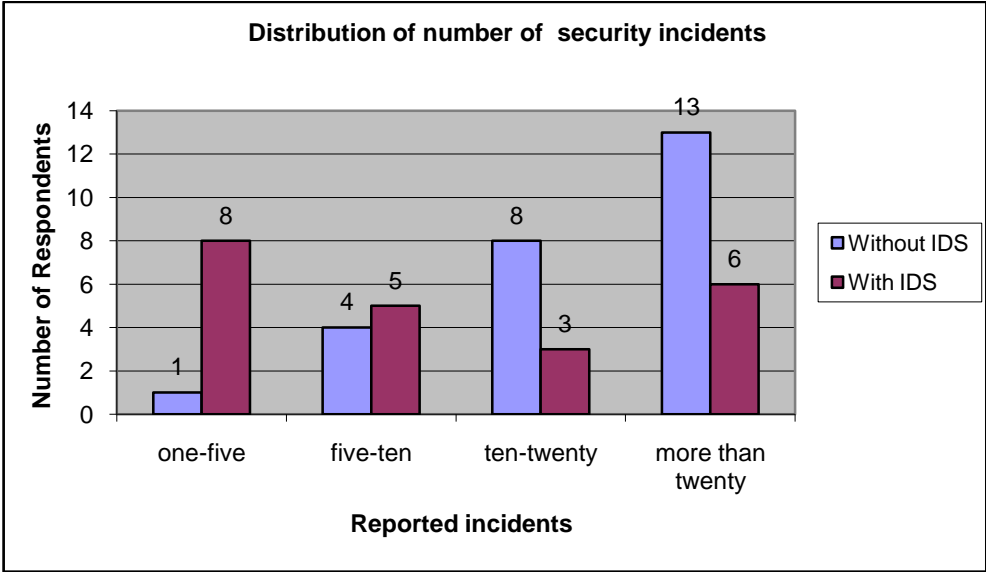


Figure 4.7: Security incident distribution with and without an IDS

Despite the conservatism of the corporate world towards the new Intrusion Detection technology, an important observation of this survey is that the **employment of IDS infrastructures relates to a smaller number of reported security incidents inside the respondents' organisations**. The graph of Figure 4.7 illustrates the distribution trend of security incidents amongst the correspondents of the survey, with and without IDS employments. It was created by combining data from questions 6 and 8 of the survey.

We can clearly observe a great reduction in the number of respondents that reported higher number of security incidents (more than 10) with the deployment of Intrusion Detection Systems. The observed distribution indicates clearly that a combination of an IDS and traditional data security technologies produces configurations susceptible to a smaller number of security incidents, enhancing the defensive effectiveness of an IT security infrastructure. This, in turn, indicates that IDS technology is a good thematic area to concentrate the efforts for insider threat mitigation.

On the other hand, it could be argued that the number of reported security incidents does not necessarily reveal their seriousness. This is true and hence latter paragraphs of this chapter provide more information about the nature and the real impact of the reported cases.

4.3 Reported Incident analysis: types, places and effectiveness of security tools

One of the main goals of the Insider Misuse Survey is to reveal the true magnitude of Insider misuse incident occurrence and compare it to that of external misuse activities. Whilst the previous sections of this Chapter focused on data originating mainly from the first part (A) of the survey, it is now time to focus on its second and third part, in order to provide useful insights into the insider misuse problem.

The majority of reported incidents in the Insider Misuse Survey appear to have internal origin.

Seventy per cent of correspondents (35 out of 50) have traced the majority of security incidents back to legitimate users, whereas less than a quarter of the participants reported attacks that were mostly related to external activities.

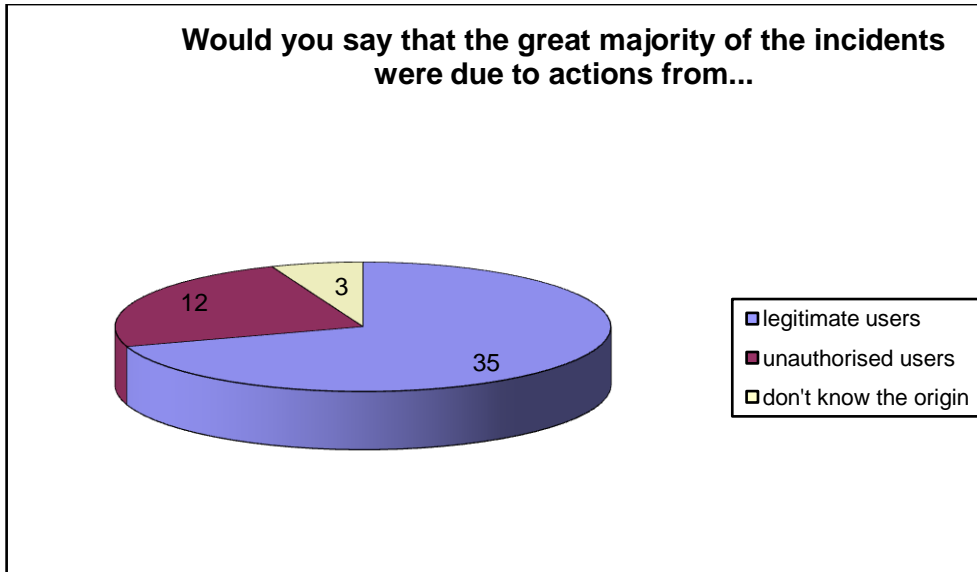


Figure 4.8: The balance between incidents of internal and external origin

The remaining six per cent (3 out of 50) were not certain about the origin of the majority of the reported security incidents. An interesting observation indicates that although all of them have employed traditional data security technologies, only one of them had employed an Intrusion Detection System. Since the number of respondents that meet this condition is relatively small and the details of their IT infrastructure configuration are not known, definite conclusions cannot be made as to why they were not able to trace the origin of these incidents. However, the result shows clearly a trend that indicates a direct relationship between the employment of an IDS and the ability to trace back incidents. This property is one of the fundamental functions of an IDS infrastructure, as mentioned in Chapter 2 of the thesis.

We asked all respondents (not only those who traced the majority of their incidents back to internal origins) to select the most likely type of internal abuse from a pre-selected list of incidents. Question 18 of the survey “Based on the experience you gained from the occurred insider incidents, which of the following types of IT misuse incidents do you think that an insider is most likely to initiate?” gave the results presented in Figure 4.9.

40 per cent (20 reported cases) of the respondents considered the storage and dissemination of pornographic material on computer equipment as the most frequent type of legitimate user misuse. This was followed by 12 reported cases of theft or fraudulent alteration of proprietary and commercially sensitive information (24%), whereas e-mail abuse was the third most common type of misusing an IT

infrastructure accounting for 16% of the reported cases. Internal virus outbreaks (two intentional incidents were recorded of which one of them was classified as intentional one), physical destruction of computer equipment (vandalism) and installation of illegal (unauthorised or pirate software packages) were encountered less often and accounted for the remaining 20% of the insider incidents.

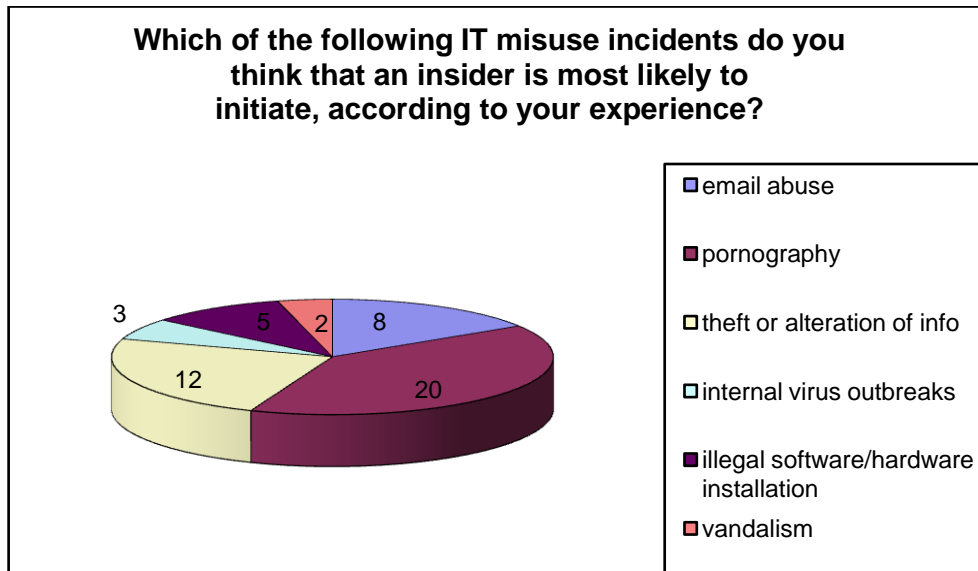


Figure 4.9: Most frequent occurring types of insider IT misuse

Although the previous statistics give an accurate picture about the range of organisations affected by insider incidents, it does not necessarily reveal their true consequences for the respondents. In addition to the frequency of occurrence, one has to consider the financial consequences resulting from these types of cases.

Figures 4.10 and 4.11 illustrate the reported substantial revenue loss for incidents of internal and external origin respectively. It should be noted that what is represented here does not constitute information based on stated sums of money. The figures represent the statistics of how many respondents admitted substantial revenue loss. A total of 34% (12 out of 35) of the respondents that have faced mostly internal security incidents reported serious revenue loss. The percentage is marginally equal to the respective figures for substantial revenue loss for a majority of external misuse incidents (33% or 4 out of 12), although the reported number of external cases was smaller than the internal ones.

More useful conclusions could be deducted if the reported lost revenue was quantified in relation to the annual turnover of the organisations. Someone could then compare properly the financial impact of external versus internal incidents. Unfortunately, only four out of the fifty respondents were able to provide an estimated amount of lost revenue. It is reasonable to assume that this level of response does not provide enough data for discovering trends and publishing useful information. Amongst other things, we chose to leave the quotation of this amount as an optional part of the survey. The provision of this kind of information can be time consuming and more importantly is a very sensitive internal issue for many organisations. Consequently, the support of important conclusions on ‘modest’ financial implication estimations would be a strategic mistake for the validity of the data and could potentially deter respondents from completing the survey.

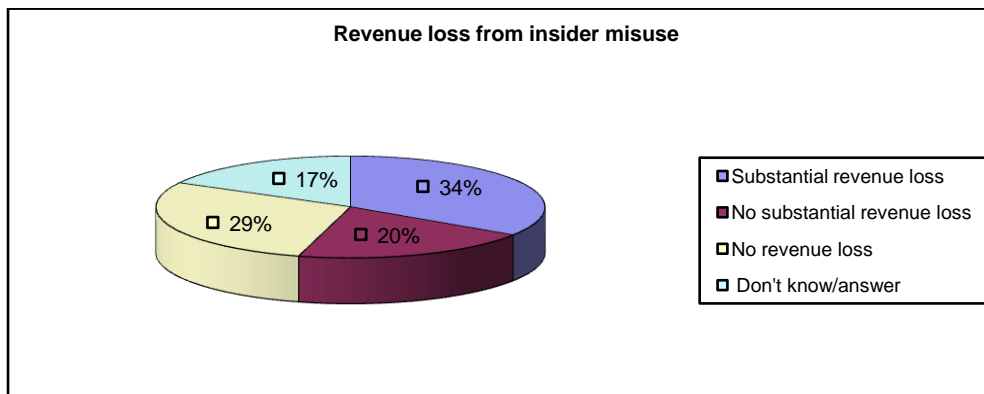


Figure 4.10: Lost revenue implications for internal incidents

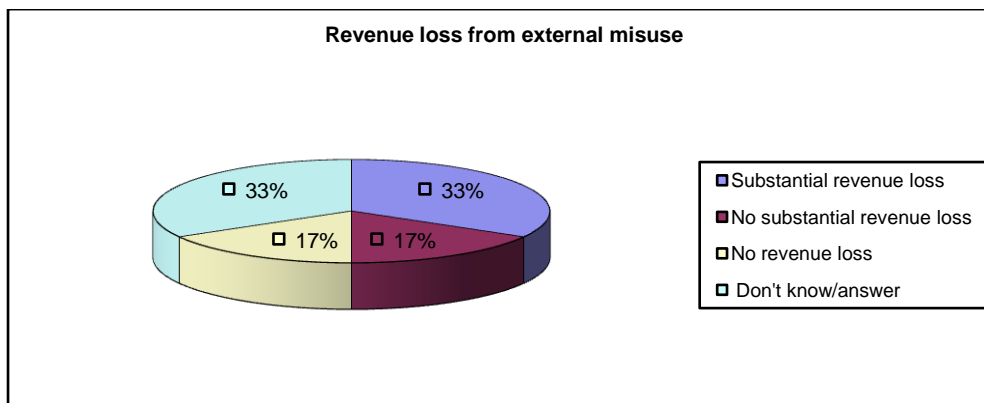


Figure 4.11: Lost revenue implications for external incidents

Another criterion that could be used to help us estimate the seriousness of insider incidents is the existence of disciplinary procedures against the legitimate users. Thus, it is reasonable to assume that the most serious of the internal misuse incidents are the ones that both contain elements of financial and disciplinary implications.

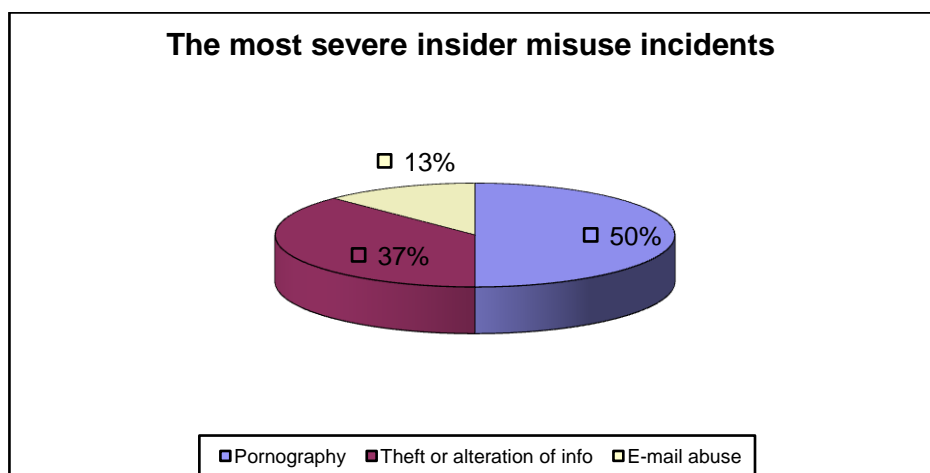


Figure 4.12: The most severe insider misuse cases amongst the respondents

Questions 12 and 13 of the survey probed for legal proceedings associated with insider incidents. The earlier question prompted the respondents to state whether their organisations faced legal action as a result of an insider IT misuse incident. The latter one asked the respondents if they thought that they should prosecute the malicious insiders. Based on that principle, we can now sort the various types of insider misuse incidents, according to a more representative level of severity, as shown in Figure 4.12. This sorting represents a total of eight incidents that were a direct result of legitimate user activities. On the contrary, there was only one reported incident of external origin with equally serious consequences. This latter fact tells us that insider misuse incidents are not only more frequent but more severe in terms of their impact for the respondents of this survey.

4.4 The magnitude of accidental insider misuse

The insider incident case studies that mentioned in earlier sections (Chapter 3) of the thesis were mostly concerned with legitimate users that intentionally misused the systems. However, it was then explained that accidental cases could not really be ruled out. The limited (for this issue) data from this survey back up this claim and reveal certain issues that deserve special consideration.

Although the number of reported legitimate accidental misuse acts was small (three out of 35 respondents that faced insider incidents) and there was no distinct pattern of emerging insider misuse cases amongst the various IT sectors, two out of the three recorded misuse acts resulted in substantial revenue loss for the affected organisations. In addition, all three organisations had deployed an IT

infrastructure that employed both traditional IT security tools and Intrusion Detection Systems, yet they were unable to prevent those incidents.

Due to the fact that the number of accidental insider misuse incidents was small, definite patterns about their severity and frequency cannot be safely established from the data of this survey.

4.5 The profile of an insider misuse act according to the respondents

The third part of the Insider Misuse Survey focused on another important aspect of tackling the insider misuse problem: that of discovering the generic characteristics of a legitimate user that misuses a system.

Earlier in the thesis, we discussed the subjective nature of the insider misuse problem in relation to the various different IT security policies. An important characteristic of the insider misuse problem is to show which things could be considered as misuse acts amongst the various IT sectors.

Question 14 instructed the respondents to choose from a range of well known legitimate user scenarios and characterise them as IT misuse acts, according to the Information Security Policy of their organisation.

Lost productivity from unauthorised use of computing resources during office hours (job and general internet browsing, computer game playing), has been a concern for many organisations [61]. The survey results reflected this problem. 23 out of the 50 respondents (46%) have identified that the extensive unauthorised usage of computer resources for non-job related purposes is considered as a misuse act by their IT usage policy. Those respondents came mainly from commercial organisations, where efficient revenue generation is a primary concern.

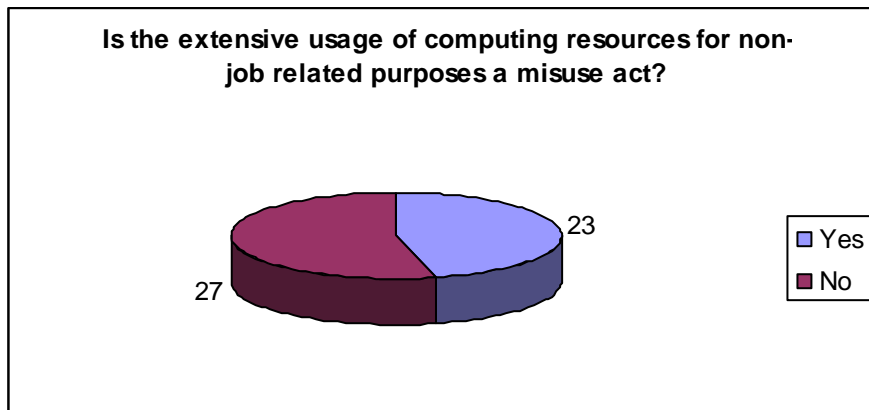


Figure 4.13: Lost productivity as a misuse act

Another popular category of classified internal misuse acts is the load that they impose on IT infrastructures. Employees tend to sometimes overuse these resources (even when they employ them for work-related purposes) causing a variety of administrative issues that could potentially result in service degradation or failure. Excessive usage of hard disk space and network bandwidth are two great examples that have forced system administrators to use mechanisms such as disk quotas [62] and network bandwidth shaping techniques [63]. The earlier is a result of the ever increasing storage needs of users, whereas the latter a consequence of the usage of many Peer-to-Peer applications (P2P).

22 out of 50 respondents of this survey have identified excessive resource utilisation as a misuse act against their IT infrastructure. The available data showed no distinct pattern that could relate this classification to a particular type of organisation. However, most of the respondents that considered excessive resource usage as a misuse act had an IT infrastructure consisting of at least 500 hosts. Therefore, it is safe to conclude that insiders are more likely to get penalised in large IT environments, where the impact of resource over-utilisation becomes more apparent.

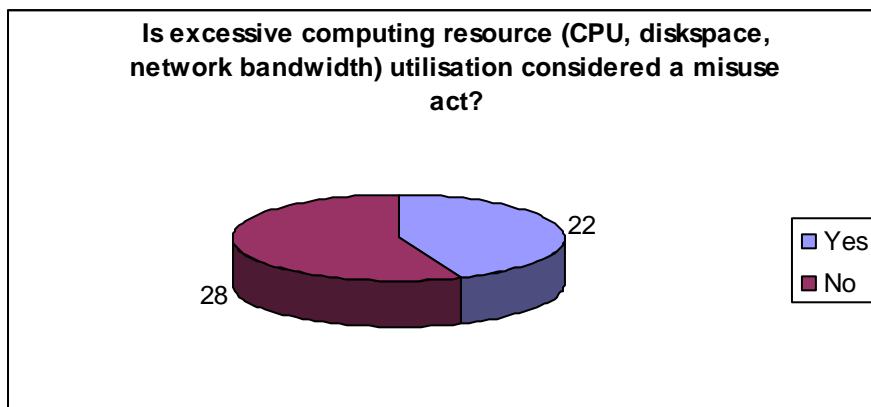


Figure 4.14: Excessive computing resource utilisation as an insider misuse act

Finally, the majority of the respondents claimed that an attempt to install one or more unauthorised applications is also classified as a misuse act for their organisations (38 out of 50 respondents). This could be used as a strong criterion for the purposes of gauging insider threats in an IT environment.

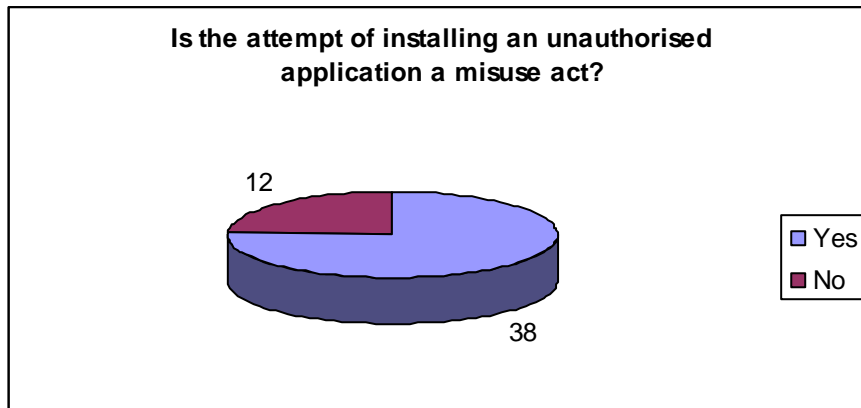


Figure 4.15: Attempts to install unauthorised applications as insider misuse acts

Although previous paragraphs have shown the variability of what normally constitutes an IT misuse act amongst the various IT organisations, there are also notable generic traits for the profile of an insider. In particular, 86% of the survey's respondents believe that sophisticated (in terms of IT system knowledge) users are more likely to misuse an IT infrastructure than less knowledgeable users. The rest (14%) of the participants supported the opinion that user sophistication is not an important insider threat indicator.

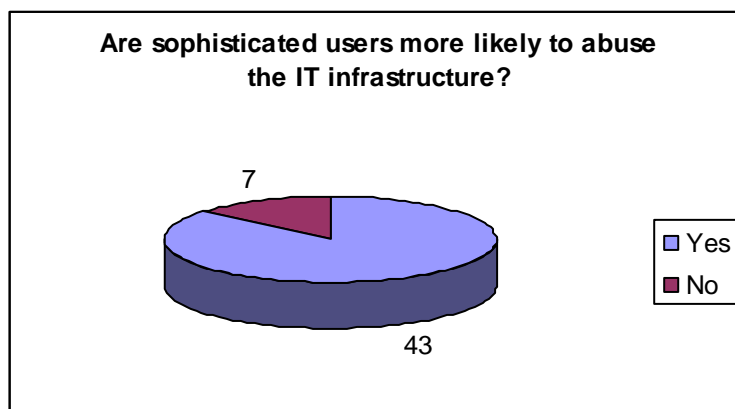


Figure 4.16: Association of user sophistication and probability of misuse amongst the respondents

In response to question 15 of the survey, "If you were designing a security pre-employment screening procedure for candidate employees, what would you think is the most important piece of information that should be included in the screening policy?" 40% of the respondents chose the verification of the reasons for leaving previous employment as the most important parameter. The verification of the

knowledge of IT security skills, previous credit difficulties as well as the existence of previous criminal conviction records were also chosen as most important parameters at a smaller scale, as shown in Figure 4.17.

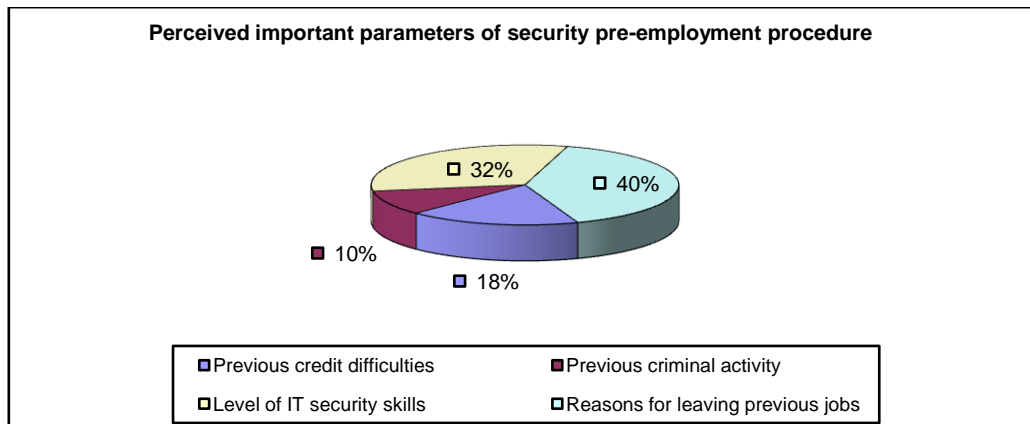


Figure 4.17: Important security pre-employment procedure parameters (the insiders past)

A final important point provides a useful technical insight on how the insider misuse acts could be traced accurately and conveniently (in terms of technical feasibility) in an IT infrastructure. Most respondents flag the examination of bespoke security tool log files as the most reliable way of tracing back the origins of an internal misuse act. The screening of web page and e-mail content followed as the second and third most important ways that a system administrator respectively. Finally, network traffic was the fourth most popular way of sensing for insider misuse activities.

The survey had offered OS log files as an option, however all the respondents turned down that option and that was expected. If external entities have the ability to cover their trails by using special audit log modification software [64], this would certainly be achievable by an insider whose access to the OS log files might be given by default or might be easier to obtain.

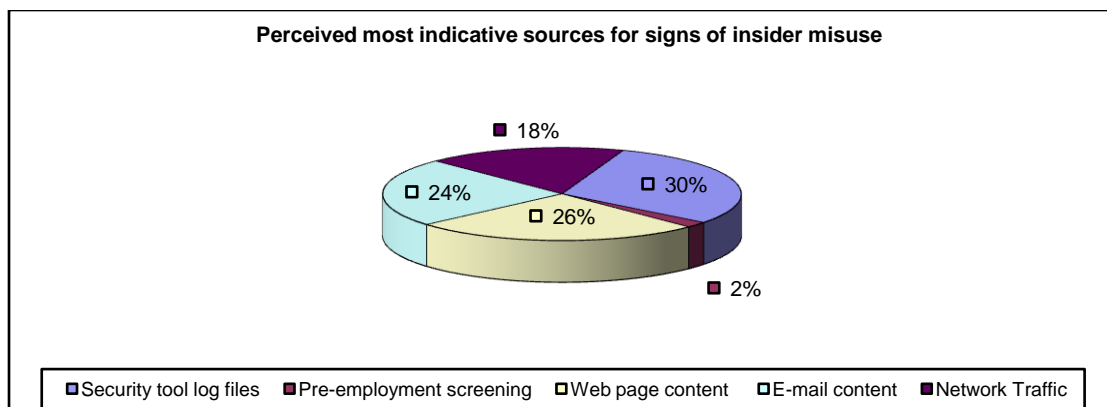


Figure 4.18: What is the most indicative source for tracing insider misuse incidents

4.6 Conclusions

Despite the small number of respondents, the statistics of the Insider Misuse Survey have clearly shown notable trends, in order to establish a profile of the legitimate employee that misuses certain elements of the IT infrastructure. The results were derived by examining the opinions of mostly technical personnel. The majority of the respondents had more than five years of professional experience in their roles. Hence, it is fair to say that although the number of respondents was nearly half than what was originally expected, the respondents' opinions are of great value since they represent experienced IT professionals that serve in a variety of IT infrastructure environments.

The fact that 35 out of the 50 respondents traced the origin of the majority of their security incidents back to internal factors reveals that **the insider IT misuse problem is certainly a well-established threat factor for the health of computing environments**. Due to the relatively small number of respondents and the fact that the thematic area of the survey was biased towards insider misuse, it is not safe to assume that the survey's majority of insider incidents can be considered a statistically safe result. However, the goal of this survey was to reveal the nature of insider incidents. The survey did not intend to prove (or disprove) the conventional wisdom of some Information Security surveys, which dictates that most security incidents occur from legitimate users. This is also the conclusion of the 2003 CSI/FBI computer crime and security survey [53].

The survey has shown that a legitimate employee is more likely to misuse the IT infrastructure by storing and viewing pornographic material in the IT infrastructure. He/she might also attempt to steal or modify (in a fraudulent way) commercially sensitive information. In this latter case, an insider misuser is likely to execute or attempt to install unauthorised software. Finally, the third most likely offensive act originating from a legitimate user might be the inappropriate use of electronic mail facilities in order to produce unsolicited communication to other parties (inside or outside) the organisation. Other offensive acts such as computer virus implantation, physical vandalism of computing equipment are less frequent. The profile of an insider threat was also refined by indicating that sophisticated users are more likely to misuse an IT infrastructure than less IT-literate users. Finally, most professionals believe that a legitimate employee's reasons for leaving previous jobs should be clarified.

This survey also revealed less frequently perceived ways of misusing an IT infrastructure, with insiders that use the computing resources in non productive ways or over-utilising them to the extent they might introduce service degradation or even service failure. What is also worth noting is that the extent to which these less conventional situations are considered as a misuse act greatly varies amongst different types of organisations. **These points were not shown clearly in previously mentioned surveys and they clearly indicate the complexity of the insider IT misuse problem.**

However, the survey has not managed to quantify the financial impact of the insider misuse problem. The majority of the respondents have chosen to respond to the question of whether they faced substantial revenue losses but they did not quantify their losses, in order to give a more accurate picture of the financial implications of the problem. Consequently, we do know that the majority of those who faced mostly insider problems also encountered financial implications, but we do not know their true magnitude.

Another area for which safe conclusions could not be derived was that of accidental insider misuse. Chapter 3 of the thesis argued that the borders between external and internal incidents were fuzzy. There were only three cases attributed to accidental insider misuse in this survey. The number of samples in this case was statistically insignificant to draw safe conclusions. In addition, for the number of respondents that faced external incidents, it is impossible to say whether internal users did accidentally played a substantial role in the case.

Nevertheless, the insider profiling information derived from this survey is a useful insight into the nature of the insider IT misuse problem and an important milestone for the thesis. At the time of writing, there was no other publicly known source that could provide data with similar level of relevance to the insider IT misuse domain as this effort did. These data are going to be used in the following chapters, in order to build the foundations of the Insider Threat Prediction Architecture.

CHAPTER 5

A TAXONOMY OF INSIDER THREAT PREDICTION EVENTS

After the detailed definition of the Insider Misuse problem and a systematic examination of its magnitude and level of severity (Chapter 4), this Chapter will take the research project one step further by proposing a bespoke taxonomy of Insider Threat Prediction factors. The specification of such a taxonomy is an important milestone for this project because it will enhance the ability to examine the problem in a more systematic way and will eventually contribute to the establishment of an Insider Threat Prediction Model (ITPM). The derivation of this model will contribute to further Insider Misuse research and development efforts around the world. At the time of writing, there are no known tools that systematically estimate the level of insider threat.

However, the derivation of a suitable ITPM scheme requires a structured approach that will identify a suitable set of threat indicator factors and provide a suitable function that quantifies them. Latter paragraphs of this chapter will present and criticize relevant research efforts in the legitimate user misuse classification. It is important to prove that none of them meets the needs of systematically performing Insider Threat prediction.

5.1 An overview and critique of existing Intrusion Specification Taxonomies

The Intrusion Detection Systems research community has developed various approaches for systematically classifying intrusion incidents. Legitimate user misuse is considered a special case of an intrusive activity. Hence, it is useful to review and criticize existing intrusion classification approaches. Furnell et al [65] provide an overview of these research efforts. In particular, there are three widely recognised Intrusion Taxonomies:

- **SRI Neumann-Parker Taxonomy [66]:** Peter Neumann and Donn Parker developed an intrusion taxonomy based on a large number of incidents reported to the Internet risks forum. The taxonomy classifies intrusions into nine categories, according to key elements that might indicate a particular type of incident. Figure 5.1 summarises the overall scheme.
- **Lindqvist and Jonssen's intrusion taxonomy [67]:** This effort could be considered as an extension of the SRI Neumann-Parker taxonomy. It further refines security incidents into

intrusions, attacks and breaches. It examines these issues from a system-owner point of view, based on a number of laboratory experiments. The results of these experiments indicated a need for further subdivision of the Neumann-Parker classes 5, 6 and 7, as shown in the second table of Figure 5.1. Their work provides further insight into the process of spotting aspects of system elements that might indicate an intrusion.

- **John Howard's security incident analysis [68]:** Largely driven from empirical conclusions, this PhD thesis study is focused on the method of attack, rather than classification categories. It establishes a link through the operational sequence of tools, access, and results that connects the attackers to their objectives.

NP 1 EXTERNAL MISUSE	Nontechnical, physically separate intrusions
NP 2 HARDWARE MISUSE	Passive or active hardware security problems
NP 3 MASQUERADING	Spoofs and Identity changes
NP 4 SUBSEQUENT MISUSE	Setting up intrusion via plants,bugs
NP 5 CONTROL BYPASS	Going around authorised protections/controls
NP 6 ACTIVE RESOURCE MISUSE	Unauthorised changing of resources
NP 7 PASSIVE RESOURCE MISUSE	Unauthorised reading of resources
NP 8 MISUSE VIA INACTION	Neglect of failure to protect a resource
NP 9 INDIRECT AID	Planning tools for misuse

Extended NP5 CONTROL BYPASS	Password attacks, spoofing privileged programs, utilizing weak authentication
Extended NP6 ACTIVE RESOURCE MISUSE	Exploitation of write permissions, resource exhaustion
Extended NP7 PASSIVE RESOURCE MISUSE	Manual browsing, automated browsing

Table 5.1: The SRI Neumann-Parker Taxonomy and its extensions by Lindqvist and Jonssen

Howards' work was one of the earliest efforts to analyse various types of intrusive activities that occurred on a wide scale. Although it cannot be considered as a pure taxonomy, the wealth of statistical analyses and the various cases mentioned provides some of the most well-written and useful material for considering/revising new taxonomies. Thus, it has historical significance as a source of systematic recording of cyber attack methodologies.

The Neumann and Parker taxonomy was the first generic systematic effort to classify intrusive activities. One interesting observation is that it is resource centric, focusing on the intrusion consequences at system level. However, the taxonomy provided unclear borders of distinction amongst the various intrusion categories. For example, if someone manages to bypass the authentication mechanisms of a system, it is not clear when the incident should be classified at NP 5 (Control Bypass), NP 3 (Masquerading) or even both. Although it is perfectly acceptable for an incident to comply with more than one taxonomy class, the whole purpose of a taxonomy is to provide a set of classification criteria that reduce the vagueness amongst the various classes to an absolute minimum. Clearly, this is not the case with certain aspects of the Neumann-Parker taxonomy.

Lindqvist and Jonssen tried to address these inaccuracies by extending the list of classification criteria with more specific taxonomy rules, focusing on the mechanism employed to achieve a successful intrusion. Although this does not constitute a radical modification of the Neumann Parker methodology, it certainly helps a person to classify an intrusive activity when the method of attack is known, reducing the ambiguous nature amongst the various NP categories.

All of the previously mentioned taxonomies describe generic intrusions, without focusing on issues that are specific to insider IT misuse. However, there have been research efforts addressing specifically the problem of legitimate user misuse, each with particular shortcomings as discussed in the following paragraphs.

Chapter 3 mentioned Anderson's [33] discussion of '**masqueraders**', '**misfeasors**' and '**clandestine**' users. However, Anderson's distinctions are considered too simplistic for the purposes of assessing insider threat: It is good to indicate the failures of authentication systems, as well as the allocation of privileges, but that is not enough information in order to classify the various ranges of legitimate user misuse acts discussed in Chapters 3 and 4 of the thesis. Moreover, Anderson's scheme contains no traces of trying to establish a clear notion of how someone can estimate Insider Threat.

A more recent and comprehensive reference to an insider taxonomy is given by Tuglular [69]. This taxonomy integrates an established security policy to the process of classifying computer misuse

incidents in three dimensions: incident, response and consequences. These dimensions can be divided into additional sub-dimensions that further classify a particular misfeasor.

Tuglular's suggested taxonomy is certainly an important step in systematising insider misuse classification. First of all, the usage of a dimension-orientated classification method is really useful, not only for controlling the granularity of information presented in each insider class, but also in developing an appropriate set of functions that systematically collect evidence for counterintelligence purposes. Tuglular introduces a complex table format containing information about an insider incident and suggests that this scheme could be best utilised when implemented with a Relational Database Management System.

Finally, Tuglular's paper is one of the first to suggest a 'target-type of threat' association as a way to prevent insider misuse. The target is an 'asset' and the rule is called a 'strategy' in the taxonomy language. The suggestion is mentioned in a single sentence and forms the basis for this research work. However, no further expansion of this concept could be found in the description of the taxonomy.

Tuglular's taxonomy is oriented towards insider incident response, rather than focusing on a set of classification criteria that could be used as threat evaluation factors. It assumes that an act of legitimate user misuse has already taken place. The goal of this project is to derive a taxonomy that relates to facts prior to the occurrence of a misuse incident.

However, the most important criticism of the previous taxonomies is not related to their potential inaccuracies or ambiguities they exhibit. Most research and development efforts in this field are at an early stage and such inaccuracies or inconsistencies are always inevitable. Although the previously mentioned taxonomies are indeed useful for the systematic study of intrusions, they offer little help to a process designed to automatically detect intrusive activities. This is because the classification criteria employed by these taxonomies cannot be qualified or quantified very easily by an Intrusion Detection System with the level of information they exhibit.

Moreover, none of these taxonomies is tailored for the process of estimating the likelihood of Insider Threat. The best way to illustrate why this is the case is by considering an intrusive scenario in the following paragraph.

A disgruntled head system administrator who has just been fired and decides to take revenge by disrupting the IT infrastructure is a typical scenario of IT misuse. As a knowledgeable insider, he/she bypasses the system authentication procedure and corrupts (and does not delete entirely) certain vital database files in order to disrupt important services. In addition, the fired system administrator also deletes the database backup copies and then covers up his actions by erasing system log files.

Although the previous taxonomies would have one or more intrusion categories that could characterise the entire incident, none of those categories could be important information for an IDS engine. The fact that an authentication procedure was bypassed (NP5 in the Neumann Parker taxonomy) and there was an active resource misuse (NP6) does not say a lot about the true intentions of the insider. Tuglular's taxonomy could also classify the incident according to the target (database files) but again that information could not be exploited fully by the IDS engine, unless more specific information about the exact nature of the file modifications is given.

If someone wants to use an IDS to detect and predict the previously mentioned activity, one has to represent events at a more system-specific level, looking at the various individual actions that achieved the result. The next section is concerned with the derivation of a suitable intrusion taxonomy scheme, in order to achieve this goal.

5.2 A proposed Taxonomy of Insider Misuse Threat Prediction Factors

The best way of enhancing the expressiveness of an intrusion taxonomy scheme for insider misuse activities is to focus on the human actions and how their consequences impact the elements of the IT infrastructure that are being targeted. The idea is that it is easier to detect which particular element is affected by a potentially intrusive action, rather than focusing on the task of sensing the origin, entity or the motives for initialising an attack.

Another important property of a suitable Insider IT misuse prediction taxonomy is the freedom of the security architect to choose what can be considered as an Insider IT misuse threat indicator. Most taxonomies enforce a rigid framework for classifying phenomena with clear borders of distinction that offer little space for subjective or varying interpretation of facts. This schema does not fit the case of Insider IT misuse prediction. Chapters 3 and 4 indicated that there are different views for what is considered as legitimate user misuse amongst the various organisations. Consequently, there are also different views for what is perceived as a legitimate user prediction threat indicator and a taxonomy tailored for the needs of a threat prediction process should be flexible enough to accommodate this fact.

A suitable Insider Misuse taxonomy scheme was presented by Furnell et al [70]. One then has to consider this taxonomy carefully and then modify it appropriately, in order to introduce the classified IT misuse prediction factors.

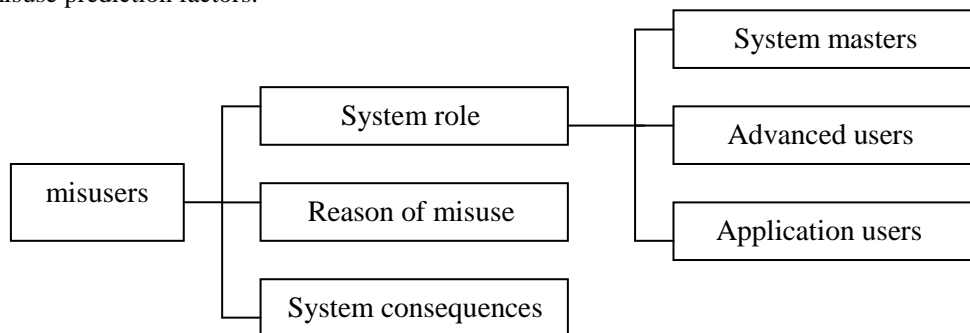


Figure 5.1: Misuser classification by system role

The human centric element of the infrastructure is justified by the fact that it is people who design, use and attack the systems [71]. There are also other factors that influence the nature of an IT misuse act, such as the derivation and enforcement of a suitable information security policy and the level of technological complexity employed inside a corporate infrastructure. Nevertheless, all actions that

constitute IT misuse lead back to human factors. Thus, a fundamental aspect of an insider misuse taxonomy should be the classification of people in three basic dimensions: **system role, reason of misuse** and **system consequences** (Figure 5.1).

'System role' is concerned with the actual (or perceived) role of a particular person with reference to a specific computer system (workstation, server, telecommunication system). The basic criterion for classifying persons in the system role dimension is the type and level of system knowledge they possess. Earlier chapters of the thesis argued that insiders constitute a greater level of threat than outsiders because of the greater level of knowledge they possess about critical components of the IT infrastructure. We have also seen from the Insider Misuse Survey (Chapter 4) that the respondents perceived that the level of user sophistication can be an important indicator of potential insider threat. Hence, it makes sense to use the level and type of knowledge of a particular legitimate user as a threat estimation criterion. As a result, we classify insiders in three basic classes:

- **System masters:** This class includes all legitimate users of the system that have full administrative privileges to the majority of the system resources and they clearly have excellent knowledge of various IT infrastructure components. Typical examples are head system and network administrators. This category of legitimate users poses a substantial level of threat to a corporate infrastructure because of the increased level of access and trust they are given.
- **Advanced users:** This sub-dimension includes all legitimate users of the system that have not got increased administrative privileges but do possess a substantial knowledge of the system internals. Application and system programmers, database administrators, as well as previous system masters and current shift operators belong to this category. These people are also very likely to misuse computer systems. Although they do not have access to a large number of system resources, they are aware of potential system vulnerabilities.
- **Application users:** This includes the rest of the system legitimate users that utilise certain standard applications, such as World-Wide-Web (WWW) browsing, e-mail and database clients. They usually have no additional access to resources, other than the ones required to run their application.

Another important factor that characterises the nature of insider misuse incidents is the reason they occur (reason of misuse). On the basis of this thought, misfeasor acts can be divided into two large categories: intentional and accidental. This classification is also employed by [33], emphasizing the importance of considering unintentional misuse incidents as equally important threats to accidental ones

Intentional misfeasor cases are performed for a variety of reasons. The best way to sub-divide them is to consider the motives in a way that could detect the ultimate goal of the abuser. It might be inferred, for example, that a legitimate user is trying to access or maliciously modify important data (data theft and data alteration), take revenge against a particular person or an entire organisation (personal differences), or deliberately ignore a particular regulation of the information security policy. The latter sub-dimension includes all goals that have not been stated and acts as a mechanism of expanding/matching the suggested taxonomy to a specific information security policy.

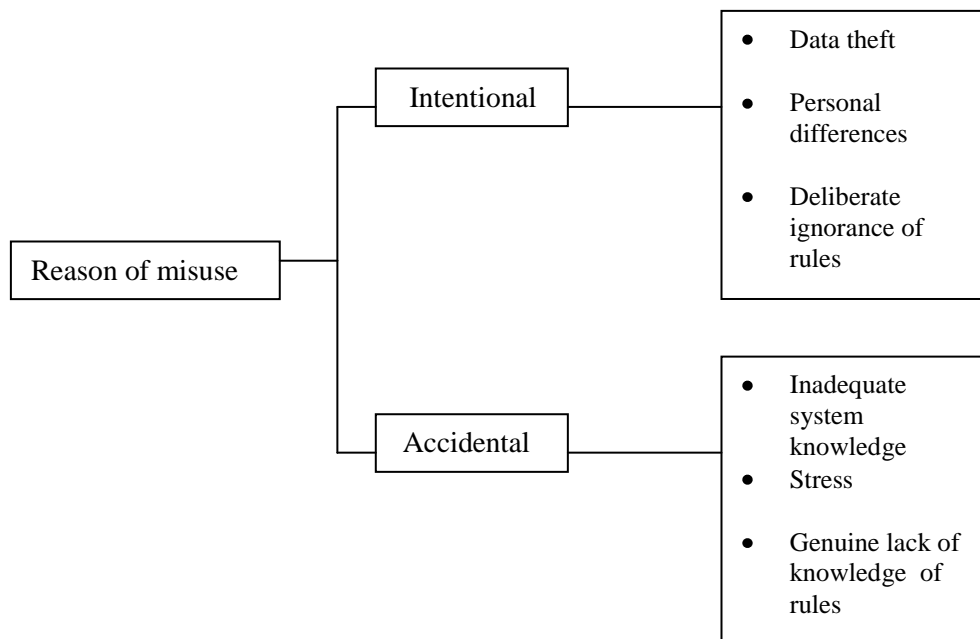


Figure 5.2: Classification of misusers by reason

On the other hand, accidental computer system misuse can be further categorised according to the real reasons that led the legitimate user to the wrong action. Issues such as inadequate knowledge of the system (due to lack of training for example), factors that can affect work-related performance (excessive workload, emotional problems) have not been addressed adequately and constitute a fruitful

area of research. Finally, it is possible that a user is unaware of a particular regulation of the information security policy. Figure 5.2 illustrates these concepts.

However, it is difficult to deduce an automated process that can distinguish between what happened by accident and what took place intentionally. For this reason, the last dimension of our classification ('system consequences') is concerned with the way a misuse act is manifested at system level.

The classification of these consequences forms a very important foundation for the Insider Threat Prediction Tool because it will be the basis for the establishment of its monitoring. It is also greatly influenced by the generic architecture of a computer system. This influence is based on the following rationale: There is a plethora of criteria that could be applied in order to evaluate insider threat. For example, social engineering and pre-employment screening procedures (the latter was indicated by the Insider Misuse Survey) might provide valuable information about the motives and the nature of the misfeasor. Someone could argue that it is possible for a human resources officer to observe the social connections of particular individuals, acting as safety measures that would flag suspicious events.

However, this type of information is often subjective- thus error prone- as well as difficult to qualify. Hence, it makes sense (especially when building an automated threat prediction tool) to classify the consequences in terms of criteria that can be easily detected by an automated software process. It can, therefore, be proven that most forms of insider IT abuse (or attempt to abuse) leave certain traces in basic components of the IT infrastructure.

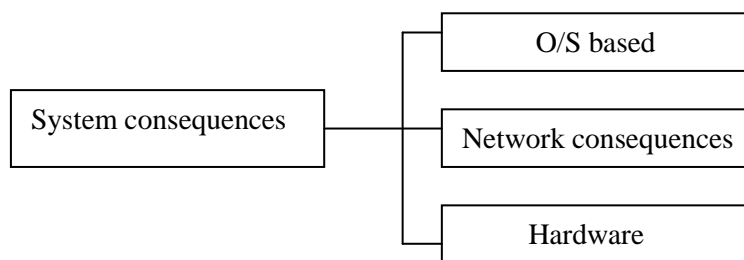


Figure 5.3: Categorisation of insider IT misuse incidents according to system consequences

As a result, there are three primary levels that address these consequences (Figure 5.3). One of them concerns issues affecting Operating System components (O/S based), the second examines threat evidence originating from network traffic (network data), and the last concerns any modifications of the physical (hardware) architecture of the system. These levels are not mutually exclusive. For

example, it is certainly possible (and common) that a particular system misuse can be traced in network data, Operating System components and hardware configuration alterations.

Bach [72] and Richter [73] are two excellent texts that describe the generic architecture of the two commercially dominant Operating Systems: The UNIX and the Microsoft Windows family of systems. Despite the substantial differences in the philosophy of their design, it is interesting to note that the core modules of a UNIX or Windows kernel provide (amongst others) two important functions: **filesystem** and **memory management**. A large number of security faults [74] involve filesystem and memory management issues. Hence, it is safe to assume that these two kernel functional attributes can be used as a strong criterion for further classifying legitimate user activities.

Figure 5.4 illustrates the File-system manipulation related hierarchy of insider misuse actions. At File/Directory level, a misuser may attempt to read or alter (write/create) certain files. These files might contain sensitive or unauthorised information (information theft or fraudulent modification of vital information).

A knowledgeable insider might also attempt to read or modify file information that is not directly related to its content. Bach and Richter emphasize that most Operating Systems allow a file to contain additional information such as access/creation/modification times as well as information that relates the file to its owner and permits access to it under certain conditions. Although the mechanisms that implement these file attributes are different amongst Operating Systems, they are collectively known as **file metadata** and they are vital mechanisms to secure the privacy, availability and integrity of the file contents. Consequently, they are good candidates for exploitation by a legitimate user who is about to perform a deliberate or accidental misuse act. The previously discussed Leeson Iguchi case is a classic example of intentional alteration of vital database files.

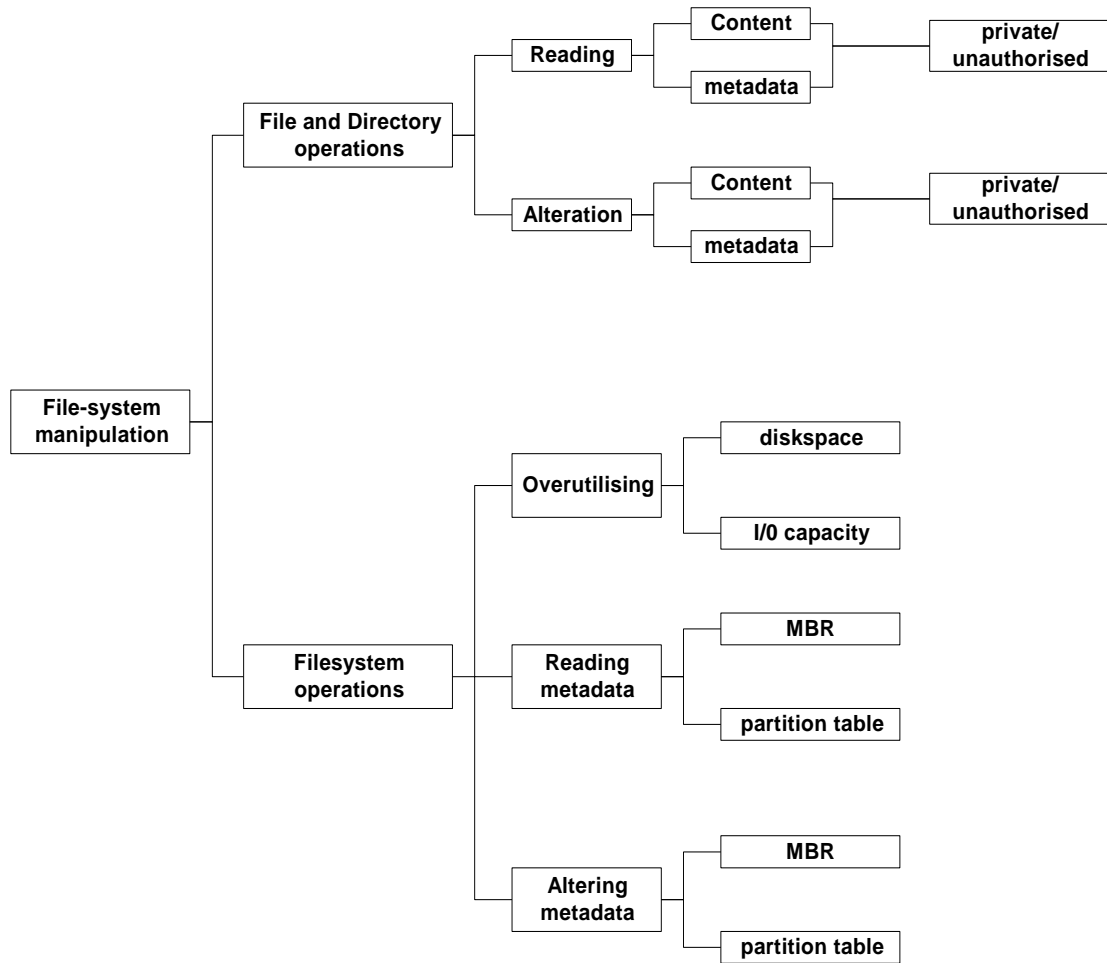


Figure 5.4: Insider Misuse Incidents classified by Filesystem manipulation OS consequences

The points mentioned in the previous paragraph are also valid for ‘filesystem’ related data. Every Operating System organises its files and directories by means of a specific set of rules that define how a file (contents and metadata) are about to be stored on the physical medium. The Operating System sub-modules that handle these issues are known as **filesystems**. Attempts to read or alter the physical medium’s Master Boot Record (MBR), intentional or accidental modification of partition table data are some of the most notable auditable actions that could point to legitimate user misuse acts. Robert Hanssen’s case is a classic reminder of this kind of activity. His specially modified 40-track floppy disk was created by a set of filesystem modification actions, in order to create a hidden area to store the sensitive information.

In addition to filesystem content and metadata modification, the Insider Misuse Survey in Chapter 4 showed that excessive disk space consumption is perceived as a problem for many of the respondents.

Under certain conditions that depend on the configuration of the IT infrastructure, a legitimate user might produce a deliberate or accidental Denial Of Service attack (DoS), either by exceeding a set of disk quota rules or running intensive filesystem Input/Output computations. At the time of writing, there were no high-profile cases documented by Information Security surveys or the mass media that fit this description. However, Appendix D contains a case study from a production-grade UNIX system, where a single malicious user managed to halt the operation of the box. In addition, the survey data together with the existence of disk quota rule mechanisms and filesystem benchmarking tools on server Operating Systems serves as a good indicator that the problem is frequently encountered in the daily operation of IT infrastructures. This is the reason why the proposed taxonomy has devoted a separate category for this type of event.

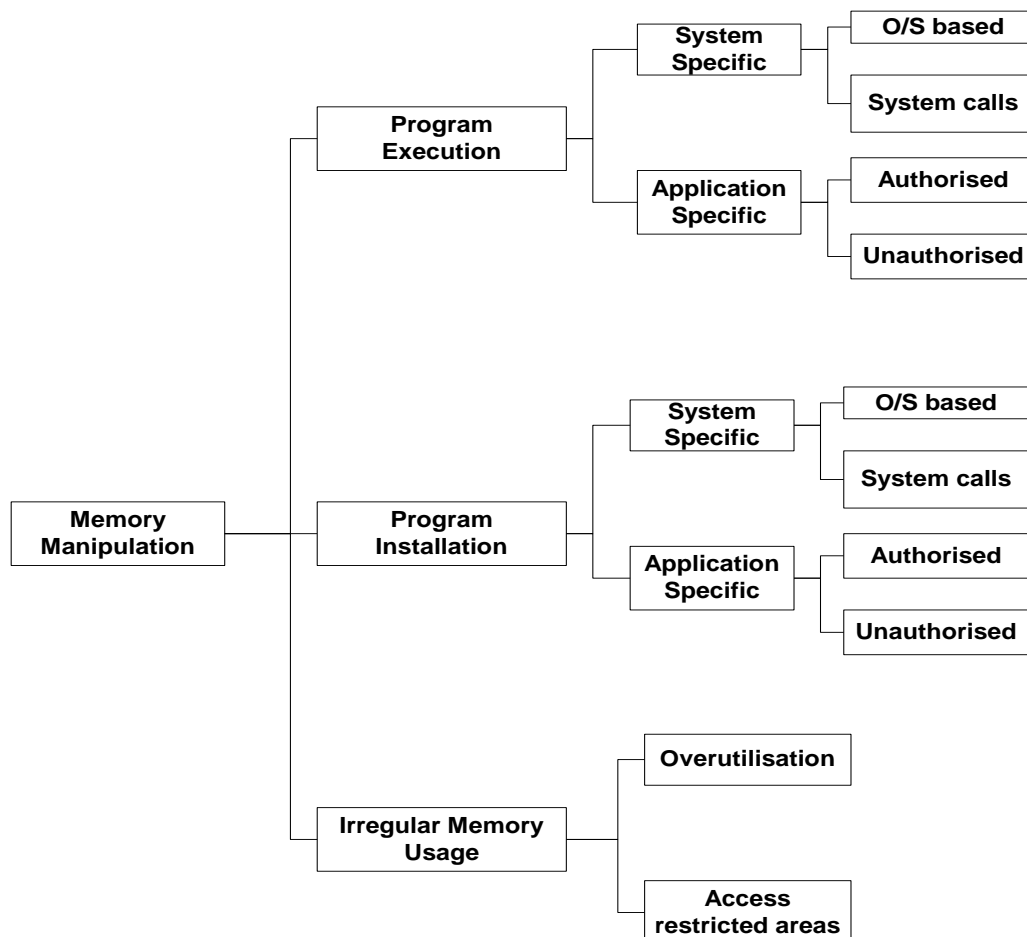


Figure 5.5: Memory manipulation OS consequences

While the filesystem provides useful insights about the actions that could indicate a potential for IT misuse acts, an equally interesting picture of insider threats could be drawn from observing the

Random Access Memory (RAM) of the system. The reason is simple. Every time an application is executed, a substantial part of its contents (program instructions together with user supplied runtime data) are transferred to RAM, where the execution of that application takes place. The 'Memory Manipulation' sub-category examines how actions related to potential misuse acts could be categorised in terms of observable system memory events (Figure 5.5).

Memory inspection is the best way to see if a legitimate user attempts to run or even install a suspicious program, a problem that was highlighted by the data of the Insider Misuse Survey. The usage of unauthorised programs is a serious issue that can also create a way for accidental misuse by introducing a number of system vulnerabilities, as described by Papadaki et al [16]. The execution or installation of these programs could be intercepted by either recognising a program's footprint in memory or by intercepting a well-known series of system calls produced by various suspicious programs. For example, the fact that a non-advanced user is trying to compile an advanced vulnerability scanning tool is an event that should be noticed and serve as a good indicator of potential misuse activities that are about to follow.

In addition, attempts to consume large memory portions of an operational system that are related to a legitimate user account can serve as good indicators of (intentional or accidental) insider misuse at Operating System level. One might argue that the 'irregular memory usage' sub-categories should really belong under the 'Program execution' hierarchy of events. However, it is possible that someone will produce a quick and easy Denial of Service attack on a running system by forcing the host to commit large portions of system memory to a process, as demonstrated in various case studies described in [75]. Moreover, a large category of security faults can be achieved by means of accessing normally restricted memory areas, creating what is commonly known as a "buffer overflow" attack [76]. As a result of these issues, it was felt that a separate sub-category hierarchy should exist to describe these events.

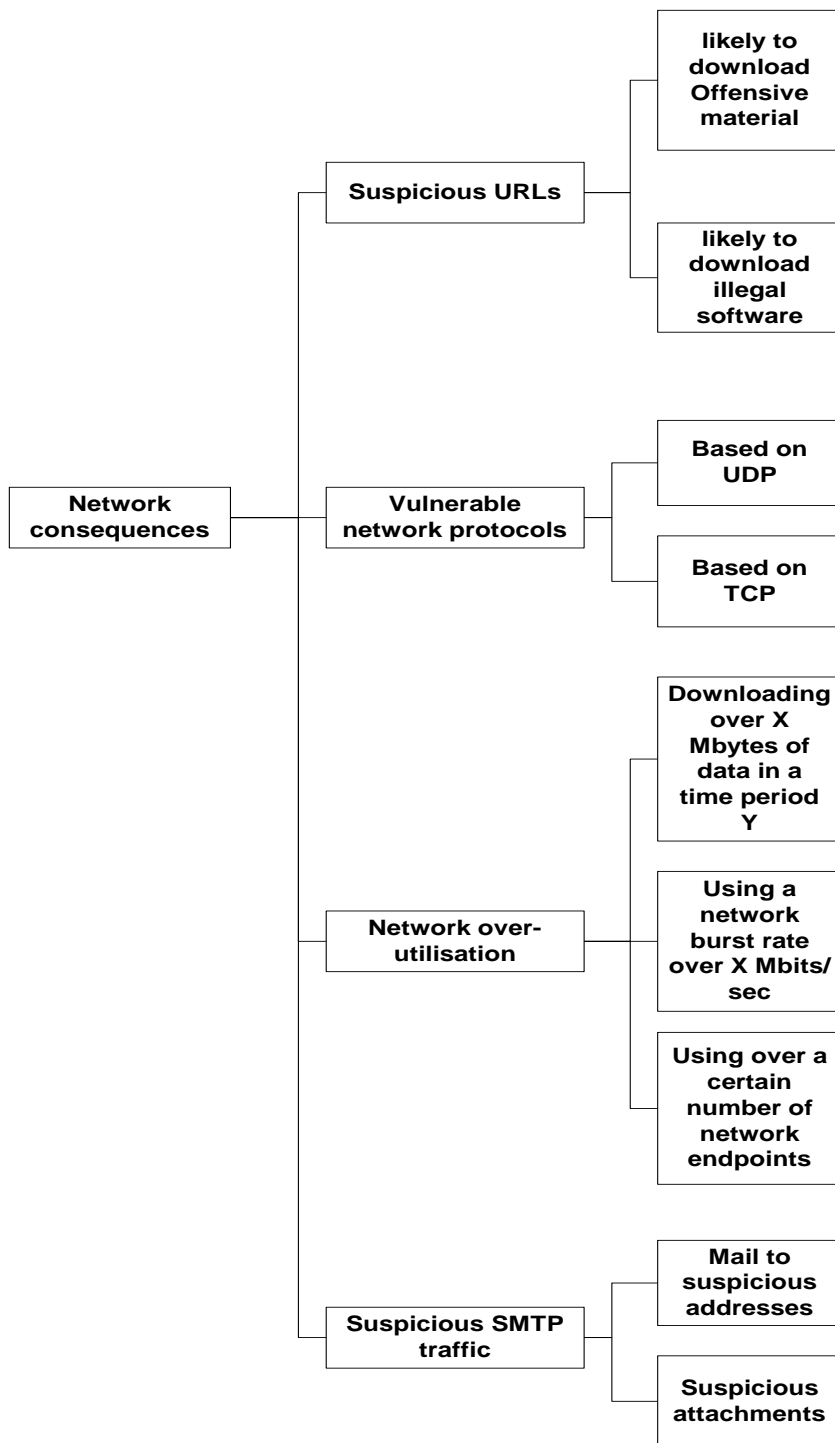


Figure 5.6: Insider Threat Prediction Factors based on Network Consequences

Network-related operations are another distinct factor that could be taken into consideration, in order to classify insider misuse threat indicators. Figure 5.6 illustrates the network-related consequences of acts that could be used as legitimate user threat indicators.

The Insider Misuse Survey indicated that a large number of IT professionals consider web page content that a legitimate user visits as an important Threat Indication factor. Hence, it is reasonable to assume

that URLs that contain a ‘promising’ link to sexually explicit content or to illegal software downloads should be noted as distinct ways of indicating potential to misuse the system (suspicious URLs).

Network packets that are associated with certain legitimate users and indicate the usage of a variety of network protocols and applications that might introduce certain vulnerabilities are also distinct ways of accidental or intentional IT misuse. For example, it could be said that a user that utilises the TELNET [77] protocol to login to a multi-user system is more likely to have her account compromised than a user who logs in via the Secure Shell (SSH) application [78] due to the fact that the earlier application transmits the user password in clear-text form across the network, whereas the latter one encrypts it. Thus, it is true to say that the TELNET user represents a higher level of threat to the system than the SSH user.

Someone might also like to differentiate between TCP and UDP based applications/protocols. From a potential threat point of view, UDP services are less secure than TCP based ones. It is out of the scope of this thesis to discuss the reasons for deriving this conclusion. Ziegler [79] discusses in detail how UDP’s lack of flow control and state mechanisms can create various data security problems. Consequently, the distinction between the usage of UDP and TCP services can serve as a potential insider misuse threat indicator, on the basis that UDP services are more likely to be accidentally (or intentionally) abused by a legitimate user.

The Insider Misuse Survey (Chapter 4) participants indicated that resource over-utilisation is an existing issue in IT infrastructures. Although the ‘Filesystem Manipulation’ subcategory of the taxonomy indicates ways with which disk storage capacity can be misused, the results of over-utilisation can also affect network capacity. For instance, a legitimate user could start downloading massive quantities of data, exceeding the network bandwidth cost budget of a business (Downloading over X Mbytes of data in a period Y). The X and Y number limits can be selected by the network administrator according to the company budget requirements.

In addition, a legitimate user might also cause network congestion by exceeding the data network’s ‘burst’ or throughput capacity or exhausting the number of available network endpoints, as described in

Sharda [80]. Bandwidth hungry applications such as video streaming players and multiple data transfers can cause congestion that can severely impact the performance of a data network or affect the Quality of Service (QoS) of certain applications that require sustained data network throughput.

Finally, incoming or outgoing SMTP headers or attachments might indicate activity related to e-mail misuse that can certainly be traced in network or host level. Outgoing e-mails that contain a set of particular files as attachments (password database files, other sensitive material) and have unusual destination addresses (unknown hotmail accounts, a large number of recipients) should serve not necessarily as intrusion indicators but as insider threat estimators.

The last system consequences category (“hardware”) plays an important role in preventing a number of computer system threats. Insiders can often access the physical hardware of the machine very easily. Thus, removal or addition of hardware components, as well as modifications of their default configuration are some important events that may act as important indicators of insider misuse prediction in a computer system.

5.3 Conclusions

This Chapter introduced a suitable taxonomy of Insider Threat Prediction Factors, based on system-level events associated to legitimate user actions. The taxonomy is tailored to the needs of automated Insider Threat Prediction because:

- It is heavily based on factors that are easily qualified by a system.
- It is flexible enough to allow the security architect to define what is considered as a threat element. For example, he could define which user network protocols are more likely to pose a threat to the system when they are utilised by a particular legitimate user. This is a necessary requirement because what can be considered as legitimate user misuse varies amongst different organisations.

The establishment of this classification scheme paves the way for the construction of a suitable Insider Threat Prediction Model presented in the following chapter of the thesis.

CHAPTER 6

MODELING THE PROCESS OF INSIDER THREAT PREDICTION

After introducing a suitable Insider Misuse taxonomy, this Chapter presents the details of a mechanism that aims to probabilistically estimate the level of legitimate user threat. After clarifying important terminology and examining relevant research efforts, the establishment of suitable threat qualification and quantification criteria is presented. The derivation of the criteria is also supported by experiments that monitored certain aspects of the legitimate user behaviour on a production-grade computer system. This is another important milestone of the research project that provides the foundation for the process of insider threat prediction.

6.1 On model derivation methodology

This Chapter uses the term ‘model’ many times. In the field of Computing, the term ‘model’ is extensively used as part of Software Engineering practices. The aim of these practices is to make the process of producing software more efficient and reliable. However, Software Engineering modelling has many potential attributes. It is useful to clarify which of its aspects are employed in this thesis and what has been excluded.

A model is a special representation of a real-world entity as a set of attributes and functions that closely resembles its behaviour. Sommerville [81] defines a model as an “abstraction of the system being studied rather than an alternative representation of that system”. The process of abstracting a real-world entity implies that not all information about its attributes and functions is transferred into the model. Only those attributes and functions important for the study of certain aspects of the entity are considered. Consequently, the first important step of deriving an Insider Threat Prediction Model is to decide which attributes and behavioural (functional) characteristics of a legitimate user are important to the Threat Estimation Process. This will produce a set of Insider Threat Qualification Attributes (ITQAs).

The next step in the process of establishing the model is to describe how the ITQAs can be quantified, in order to estimate the level of insider threat per individual user. This will involve the establishment of

a suitable mathematical function, which will take as input a number of ITQAs and will associate them with a certain level of threat. We shall call this function the Estimated Potential Threat function, which quantifies the ITQAs.

At this point, the overall target of our model will be achieved: the establishment of a mechanism that will map ITQAs to certain threat levels. However, a formal Software Engineering modelling process does not stop here. There is a plethora of modelling techniques that can define the model's data format (semantic data modelling [81]), as well as the reliability of the model functions (formal methods [81]). The thesis has omitted those formal aspects of Software Engineering based modelling, since the goal of the research project was to produce the design for a proof-of-concept system, rather than a production grade system.

6.2 Previous Insider Threat Modeling efforts

The development of insider threat models is a relatively new idea. Wood [82] provides an excellent basis for qualifying a set of metrics to mitigate insider threat. Most of these criteria are in line with the conclusions derived by the Insider Misuse Survey, as well as issues discussed as part of the insider misuse taxonomy presentation in Chapter Five of the thesis.

In particular, Wood suggests that a malicious insider can be qualified in terms of distinct attributes:

- **Access:** The insider has unlimited access to some part or all parts of the IT infrastructure and the ability to physically access the equipment hardware. Consequently, the insider can initiate an attack without triggering traditional system security defences.
- **Knowledge:** The legitimate user is familiar with some or all the internal workings of the target systems or has the ability to obtain that knowledge without arousing suspicion.
- **Privileges:** The malicious insider should not have problems obtaining the privileges required to mount an infrastructure attack.
- **Skills:** The knowledgeable insider will always have the skills to mount an attack that is usually limited to systems that he/she is very familiar with. The model assumes that a given adversary is unlikely to attack unfamiliar targets.

- **Tactics:** This attribute refers to the methods used to launch the malicious attack. They are dependent on the goal of the attack and might include a variety of scenarios such as plant-hit-and-run, attack-and-eventually run, attack-until-caught as well as passive information extraction acts.
- **Motivation:** Insiders might launch the attack for profit or sabotaging the target organisation. Some of them might mount an attack for personal reasons such as taking revenge against the enterprise or even satisfy their plans to invoke some policy change inside an organisation.
- **Process:** The model assumes that a legitimate user follows a basic predictable process to mount an attack that consists of distinct stages. First the malicious adversary will become motivated to mount the attack. The next logical stages involve the identification of the target, the planning of the attack and finally the act of mounting the attack itself.

All of the previously mentioned attributes emphasize important aspects of the insider misuse problem. Previous Chapters of the thesis have presented comments on the importance of insider attributes such as role, knowledge and privileges. A very useful comment with respect to the Insider Threat Estimation modelling comes from the process attribute. The fact that Wood characterises an insider attack as a ‘predictable’ process is a positive sign for the goal of this project.

However, Wood’s criteria do not necessarily represent a clear picture for the establishment of an insider threat prediction model. Not all stages of an insider attack can be safely predicted. Some of the previously mentioned attributes are difficult to qualify by an Intrusion Detection System. The ‘motivation’ adversary attribute is one of them.

It is very difficult to establish a set of sensors that could reliably deduce when an individual becomes motivated to misuse a system. For instance, let us suppose that IDS sensors record that a commercially important file is transferred from a disk to an external storage medium in the early morning hours. The fact that this particular file transfer took place could be related to a malicious act or an innocent file backup process performed by the system administrator as part of a system recovery process. It is important to maintain a record of these types of events, but their existence does not necessarily indicate an insider misuse event in progress. The plethora of the potential origins of such an event would

increase the amount of information to be evaluated. Consequently, the complexity of the algorithms to capture and evaluate this type of information would deem this attribute's exploitation impractical. At the time of writing, there is not a known algorithm, which is able to capture and evaluate that kind of information in existing Intrusion Detection Systems.

If someone observes the different stages of the 'process' insider-modelling attribute, it becomes clear that the closer we get to the actual attack itself, the stronger the indicators of insider threat. Although detecting motivation might be tricky, with a carefully chosen quantification scheme of ITQAs, someone could sense an adversary during the target identification and attack planning stages.

In addition, other attributes seem to be so closely related that might be redundant. For instance, it would be more logical to combine the attributes of 'access' and 'privileges' into one 'insider access rights'. The issue of obtaining a privilege to mount an attack should include logical and physical means of interacting with the systems. The same could be said for the attributes of 'knowledge' and 'skills', because the ways in which a legitimate user gets to know a system and what can be inferred from the insider's system knowledge are issues that are closely interrelated.

Due to its introductory scope, Wood's paper [82] does not deal with the quantification of insider threat attributes. It is unknown whether this means that a suitable threat modelling function has been deduced as part of the preliminary model mentioned in this paper. The author has yet to publish a completed version of the model for verification.

A more recent research effort by Schultz [83] presents a preliminary framework for understanding and predicting insider attacks by providing a combination of behavioural and system usage ITQA metrics. The paper mentions the detection of system usage patterns that may act as "signatures" of a legitimate user or certain indicators of an attack preparation ("deliberate markers" and "preparatory behaviour"). Legitimate users might also make noticeable mistakes in the process of misusing a system (meaningful errors). Finally, "correlated usage patterns" refers to sequences of actions that might not be detected in individual systems but they could certainly indicate misuse when considered against multiple systems.

Schultz also suggests that certain aspects of a legitimate user's personality could serve as threat indicators. In particular, on-line (e-mail, IRC or other forms of computerised human-to-human communication) verbal behaviour with signs of aggression, dominance towards particular people might serve as a good prognosis factor of certain attacks ("verbal behaviour"). Furthermore, based on the works of Shaw et al [71], the research suggests that it is possible to examine other "personality traits" as potential threat indicators.

The Schultz preliminary framework even suggests a way to quantify all these metrics by means of a multiple regression equation that consists of the summation of the ITQA metric variables multiplied by their weightings. If $X_1, X_2, X_3 \dots X_N$ represent the quantified ITQA metrics, W_i ($i=1, i=N$) their respective weights and C an arithmetic offset constant, then the expected estimated threat X_e is derived below:

$$X_e = (\sum W_i X_i) + C = W_1 X_1 + W_2 X_2 + W_3 X_3 + \dots + W_N X_N + C$$

One notable absence of the Schultz insider threat prediction scheme is that there is no direct association between the estimated level of threat and the legitimate user's level of technical knowledge. Although the proposed metrics can provide evidence that could be used to infer the level of user sophistication, there is no mentioning of a mechanism that takes that into consideration. Given the fact that, at the time of writing, the field of Insider Threat modelling is premature to reveal any usable results, it is difficult to prove the real impact of user sophistication on the threat level. On the other hand, Wood's model, a number of case studies mentioned in Chapter 3 and the Insider Misuse survey results (Chapter 4) provide strong indications that there is a direct relationship between these two concepts. In that sense, the lack of a legitimate user sophistication gauging component could present a serious omission of the Schultz framework.

In addition, the exploitation of future mechanisms that will associate personality traits to potential misuse threat levels raises certain ethical and feasibility concerns. It is outside the scope of the thesis to examine ethical issues and the various laws that are associated with them. Nevertheless, the process of designing a model that is going to be employed in the real world should take into consideration its troublesome aspects. A metric that penalises real people in terms of their character traits will be

considered unethical by many and depending on regional legislation may be also unfeasible to implement.

In summary, the Schultz framework is more refined than Wood's earlier Insider Threat model in that it provides more concrete examples of ITQA metrics as well as a basic quantification mechanism for them. However the framework is still in its infancy. The author acknowledges that the chosen metrics need further refinement in order to prove their usefulness in a threat estimation process.

Both models concentrate on malicious (i.e. intentional activities) without considering accidental insider misuse actions. This can be a serious omission for a model that aims to address all aspects of the insider threat issue, as the problem of accidental insider misuse does exist and can have serious consequences, as shown in the Norwich Union Case [59].

Finally, all of the aforementioned research efforts do not address the issue of managing the representation of the data that feed the model component functions. One could argue that a preliminary model design needs to focus more on the scope, quality and quantity of its insider threat modelling functions. On the other hand, a well-thought definition of the procedures that represent and store the data that feed the threat modelling functions may have a notable impact on the computational efficiency and acceptance of the model. The reasons that support the need for this requirement are going to become apparent in the sections that follow, as well as in Chapter 7 of the thesis.

For all these reasons, we need a more formalised and broader model description. The next sections of this Chapter provide a detailed description of the proposed Insider Threat Prediction Model.

6.3 The Insider Threat Prediction Model

After discussing the various advantages and disadvantages of similarly minded research efforts, this section will present an Insider Threat Prediction Model that attempts to overcome the shortcomings of previous research work. A preliminary version of this model has been published by Magklaras and Furnell [70].

Considering a legitimate user population that has access to various components of an IT infrastructure, the core of the Insider Threat Prediction Model is a three-level hierarchy of mathematical functions evaluated in a bottom-up approach. At the top level, the Evaluated Potential Threat (EPT) function provides an integer value that quantifies and classifies the potential threat for each legitimate user into three different categories. If x denotes the computed EPT for a legitimate user, EPT_MAX a threshold EPT value for considering the user a threat and EPT_MIN a threshold EPT value for considering the user's on line presence as suspicious, then:

- **Important internal threat ($x \geq EPT_MAX$):** It indicates a high potential of a particular user misusing the system.
- **Suspicious ($EPT_MIN \leq x < EPT_MAX$):** This flags a condition where a particular user behaves in a manner that does not constitute a substantial threat but it is still a concern.
- **Harmless ($0 \leq x < EPT_MIN$):** To indicate that the potential of misuse is nearly non existent for a particular user.

It should be emphasized that the derived EPT value is an integer that represents a measure of the likelihood of system misuse, ranging from 0 to 100 points. Higher EPT scores indicate more probable threats. However, it should be noted that the model equations presented in this Chapter do not represent a validated probabilistic model. Since EPT represents likelihood of Insider IT misuse occurrence, one would expect the formulae to map a series of data to a probability figure. Although this is the aim of the model, in addition to the EPT function, one would then have to carefully relate the derived EPT score to the fact of whether the event really occurred or not. This comparison would facilitate the construction of proper probability distribution function, which relates a range of data to a probabilistic

value of incident occurrence. As a result, the reader should be aware that there is a difference between the EPT score and an actual probabilistic figure.

Each of the threat component functions models particular aspects of insider attributes and behavior. At the moment, in order to devise a well structured organisation of threat components, the suggestion is to provide two threat component functions. The first one considers legitimate user attributes such as access rights and professional role, whereas the second evaluates potential threat simply by examining aspects of user behavior at the system level. Figure 6.1 illustrates the proposed formula.

$$\begin{aligned} \mathbf{EPT} &= \sum \mathbf{F}_{ITPQA} \Rightarrow \\ \mathbf{EPT} &= \mathbf{F}_{attributes} + \mathbf{F}_{behavior} \Rightarrow \\ \mathbf{EPT} &= \mathbf{C}_{role} + \mathbf{F}_{accessrights} + \mathbf{F}_{behavior} \quad (1) \end{aligned}$$

Figure 6.1: The three-layer ITPM function hierarchy

It is envisaged that $F_{behavior}$ has a greater weight in the process of calculating the user EPT than $F_{attributes}$. Legitimate user attributes are important and should always be taken into consideration. However, it is expected that amongst two users that have the same attributes, it is the gauging of their behavioral characteristics that can decide which one is more likely to constitute a greater level of threat for the system. Hence, a total of 30 points will be contributed to EPT by $F_{attributes}$ and 70 points by $F_{behavior}$.

In addition, Table 6.1 lists the maximum weights of the nine top-level EPT formula components that are explained in detail in latter sections of this chapter. Some of these components are constants (Crole, Csysadm...etc) that belong to the $F_{attributes}$ function, whereas others constitute sub-functions of the $F_{behavior}$ function that address the assessment of the legitimate user on-line behavior.

The sum of the weights adds up to 100. This corresponds to a probability range of 0%-100%. The derivation of the defaults maximum values is a consequence of the aforementioned ratio between $F_{attributes}$ and $F_{behavior}$. Due to the initial choice of weights between the $F_{attributes}$ and $F_{behavior}$ functions, the 5 constants of $F_{attributes}$ have a maximum score of 6 points, contributing a total score of 30. The rest of

the EPT components, should total a score of 70 points attributed to F_{behavior} . Fsophistication attributes 10 of these 70 points and the rest of the sub-functions can score a maximum of 20 points each. Consequently, the default values preserve that ratio and attribute almost equal weights for each sub-function component.

EPT Component	Maximum Weight	Meaning
Crole	6	What is the documented role of the user inside the organization?
Csysadm	6	Has the user access to Operating System administration utilities?
Ccriticalfiles	6	Is it meant for the user to access commercially sensitive files?
Cutilities	6	Can the user execute application critical utilities?
Cphysicalaccess	6	Has the user physical access to critical parts of the IT infrastructure?
Fsophistication	10	How capable is the user in terms of his computer system knowledge?
Ffileops	20	What are the signs of forthcoming insider misuse at file-level?
Fnetops	20	What are the signs of forthcoming insider misuse at data network level?
Fexecops	20	What are the signs of forthcoming insider misuse at program execution level?

Table 6.1: EPT component Weight Matrix

It should be emphasized that the proposed maximum weights on table 6.1 are not meant to be fixed. A system administrator/security specialist can re-define the maximum weights, in order to reward a particular metric that he trusts more than the others. For this reason, the nine weights of Table 6.1 constitute the *Weight Matrix*, a very important parameter for the ITPM system. The Weight Matrix allows a specialist to further tune the sensitivity of the model, depending on the way he constructs misuse signatures, his confidence on the various metrics and the nature of the incident he is trying to predict. This feature enhances the adaptability of the proposed model scheme.

6.3.1 Modeling legitimate user attributes

The $F_{\text{attributes}}$ function examines particular user characteristics associated to their role and level of access inside the IT infrastructure. C_{role} represents an arithmetic constant associated with the role of the user in an IT environment. The Insider Misuse Prediction taxonomy of Chapter 5 (Figure 5.1) discussed three possible categories of users with respect to their IT role inside the organization: ‘System masters’, ‘advanced users’ and ‘application users’. As discussed in Chapter 5, system masters and advanced users will be more likely to misuse the IT infrastructure than application users. This fact should be reflected in the arithmetic value assigned to this constant. Thus, the following set of inequalities should always hold true:

$$C_{\text{systemmasters}} > C_{\text{advancedusers}} > C_{\text{applicationusers}}.$$

$F_{\text{accessrights}}$ is a nested function that associates threat levels to file, application and physical access rights. Earlier chapters of the thesis discussed real world cases where certain files were manipulated with certain applications. It is hence logical to assume that an important threat indicator is whether access to these files or applications is provided to a particular user by default. For example, when a legitimate user has access to a database file that contains all the infrastructure system accounts or a Word Document that contains the latest commercial secret of a company, he/she would have more chances of inflicting a serious attack than a user that does not have the authority to access these files.

The same could be said for the right to execute certain applications. A user that has access to applications such as database manipulation tools, password cracking programs or other system administration related utilities should rank high in a threat estimation process.

Finally, Chapter 5 discussed also access to physical hardware as an insider misuse threat indicator. The fact that some users can physically enter a server room where servers, backup media and network access points are located should also be taken into account by an insider threat prediction tool.

Consequently, there needs to be a direct association of files, applications and physical locations to certain levels of threat. The security administrator will have to clearly identify these points and rank them according to their level of importance. These thoughts are reflected by the following equation in Figure 6.2.

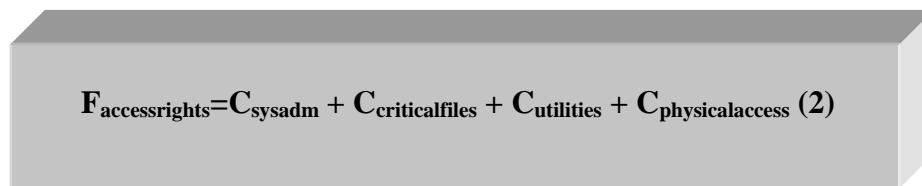

$$F_{\text{accessrights}} = C_{\text{sysadm}} + C_{\text{criticalfiles}} + C_{\text{utilities}} + C_{\text{physicalaccess}} \quad (2)$$

Figure 6.2: $F_{\text{accessrights}}$ formula

The association can be achieved by devising a set of arithmetic constants to indicate certain facts to the threat prediction process. C_{sysadm} represents the fact that a user has access to Operating System administration files (both executables and configuration files). $C_{\text{criticalfiles}}$ is set when a legitimate user has access to one or more commercially sensitive files.

$C_{\text{utilities}}$ is a constant that indicates access to administration utilities of third party applications, as opposed to C_{sysadm} which is concerned with OS level utilities. The reason this distinction is made is due to the fact that many critical applications employ their own authentication system to control certain operations, in addition to the Operating System authentication mechanisms. A characteristic example of an application that meets these criteria is a Relational Database Management System (RDBMS). For example, the MySQL™ RDBMS [84] is a popular product that employs a “root” or “Administrator” account, in order to control database contents and operations. This account is often not related to the underlying OS Administrative accounts. This means that an ordinary OS user could have full control of the Database operations. If this database controls vital data, such as a company’s funding, employee or payroll applications, access to these facilities needs to be indicated by additional flags to the threat estimation process.

Finally, the physical access part is accommodated by the last constant $C_{\text{physicalaccess}}$ that indicates physical access to critical servers, backup media locations and network access points (places where wiring panels, switches or routers are placed).

Most of the previously mentioned associations have to be done manually by the security officer or system administrator(s). As a result, the $F_{\text{accessrights}}$ function represents a relatively static element of the threat prediction process, expressing some essential initial attributes of the legitimate user. The same cannot be said about the $F_{\text{behaviour}}$ function. Designed to associate the user on-line actions with the potential of IT misuse, the character of this function is much more dynamic than the one of $F_{\text{accessrights}}$ and is presented in the next section.

6.3.2 Modeling the legitimate user behavior

Modeling the behavior of legitimate users is viewed as a process that has two distinct components. One of them relates to the technical aptitude or sophistication of the user. The Insider Misuse Survey of Chapter 4, as well as the Insider Misuse case studies of Chapter 3 indicated that user sophistication was viewed as an important indicator of potential threat. The second aspect of this modeling effort concerns what the user actually does on a live system. Chapter 5 of the thesis proposed a taxonomy for Insider Misuse Threat Prediction and concluded that misuser actions could be traced at file, memory and LAN data levels. Consequently, F_{behavior} can then be defined below.

$$\mathbf{F}_{\text{behavior}} = \mathbf{F}_{\text{sophistication}} + \mathbf{F}_{\text{fileops}} + \mathbf{F}_{\text{netops}} + \mathbf{F}_{\text{execops}} \quad (3)$$

Figure 6.3: The F_{behavior} component sub-functions

It can be argued that the technical aptitude of the user can be viewed more like an attribute rather than a behavioral characteristic and thus it should belong to the $F_{\text{attributes}}$ function. In the traditional sense of the word ‘attribute’, the competency of an individual is part of his/her attributes. However, an automated process can only determine this attribute by examining the behavioral patterns of the user.

Moreover, technical aptitude is a dynamic characteristic that evolves over time. Although the experiments of the thesis have not looked into the evolution of user sophistication over time, it is

reasonable to assume that users gain experience and hence their technical aptitude increases over time. As a result, the measurement of user sophistication does not fit the static character of the $F_{\text{attributes}}$ function. For a group of users that have the same documented role and system access rights over a period of time, the technical aptitude amongst them varies. This is shown in the sections that follow.

For these reasons, it was felt that user sophistication should be part of F_{behavior} instead of the $F_{\text{attributes}}$ function.

6.3.2.1 Modeling user sophistication

$F_{\text{sophistication}}$ provides a mechanism to profile every legitimate user in terms of his/her level of technical sophistication. The process of establishing the metrics for classifying users according to their level of technical knowledge involves gauging essential elements of their on-line behavior and then establishing a pattern that can clearly distinguish between advanced and non advanced users.

The idea of modeling end user sophistication is not a new one. Evans and Simkin [85] have produced early studies on measuring sophistication, amongst Computing Professionals and Computer Science students. Their study tried to identify how competence in Computer Programming can be correlated to factors such as age, gender and a range of other individual differences. However, their effort focused only upon computer professionals. A generic End-User Sophistication model should address a much broader user base, not only professionals and students of the IT field. Nevertheless, Evans and Simkin were one of the first to consider technical aptitude (in this case computer programming ability) as an End-User Sophistication parameter.

Huff et al [86] have systematically attempted to produce a more generic model of end user sophistication. Their paper discusses how end user sophistication could be evaluated for the purposes of increasing the efficiency of human resource management inside an organization. The scope of their work is clearly outside the field of Intrusion Detection. However, their conclusions can be utilized in order to craft suitable algorithms that gauge user sophistication.

Huff et al conducted interviews on 31 employees from 8 different organizations. The interviews had a semi-structured nature, asking the subjects to fill short questionnaires and talk about their experience on particular IT issues. The questions ranged from summarizing the software tools they use on a daily basis, how much training they had undertaken on these tools and what were the perceived difficulties they had faced with these IT applications. The results were collected and analyzed by the authors and an additional panel of Computer Science Academics.

The result of this analysis was the formulation of an ‘End User Computing (EUC)’ sophistication model that classified users in terms of three important attributes:

- **Breadth of knowledge:** Their findings indicate that advanced users were able to employ a greater variety of IT tools than intermediate or novice ones.
- **Depth of knowledge:** The level of mastery of a particular IT sub-domain or application (gained either by extensive training or hands-on experience) is proportional to the level of user sophistication.
- **Finesse:** The ability of a user to solve particular IT problems in efficient and innovative ways, given a certain level of breadth and depth capability is also an end-user sophistication classification metric.

The authors do not provide a structured methodology of how exactly they measured the ‘finesse’ attributes of users. Although the way (tools and their combination) of solving a series of problems is a reasonable metric of the end user abilities, it would be difficult to devise standardized tests for an automated IDS algorithm on a live system. Consequently, someone may focus on the breadth and depth dimensions of EUC sophistication.

Prior explaining how the aforementioned concepts could be turned into a workable model, it is important to mention for reference purposes that the experiment described in this section uses the following Weight Matrix values:

$$(6,6,6,6,6,12,18,18,20)=(C_{role},C_{data},C_{hardware},C_{sysadm},C_{utilities},F_{sophistication},F_{fileops},F_{execops},F_{netops})$$

Thus, Fsophistication contributes a maximum of 12 points to the EPT value.

In order to devise a metric for measuring the breadth of knowledge, if n represents the number of unique applications executed by a particular user per session and c the number of sampled user sessions, then:

$$\text{avdiffapps} = \sum_{i=1}^c n_i / c \quad (4)$$

Figure 6.4: Breadth of knowledge formula

This scheme will reward more points to users that execute on average a greater variety of tools. In order to dimension the avdiffapps values to fit in the proposed scales of the ITPM scoring scheme, it is necessary to consider the average values of avdiffapps for each user category. If μ represents the arithmetic average of avdiffapps for every user category, then:

$$\begin{aligned} \text{Fbreadth} &= 6, \text{ if } (\mu_{\text{ordinary}} < x \leq \mu_{\text{advanced}}) \text{ OR } (\mu_{\text{ordinary}} < x \text{ AND } x \geq \mu_{\text{advanced}}) \\ \text{Fbreadth} &= 3, \text{ if } \mu_{\text{novice}} < x \leq \mu_{\text{ordinary}} \\ \text{Fbreadth} &= 1, \text{ if } 0 < x \leq \mu_{\text{novice}} \end{aligned} \quad (5)$$

Figure 6.5: Fbreadth function based on the avdiffapps value

Huff et al [86] claim that “depth capability has much to do with mastery of the features and functions of different types of application systems, practices, techniques etc”. In order to inspect these parameters on a working system, one has to devise mechanisms for checking:

- I) The type of applications utilized on average and rate them in terms of the level of system knowledge they require in order to be used.
- II) The way each of these applications is called and used by considering issues such as way of execution (scripted versus manual) as well as number and type of arguments.

In order to realize the requirements of mechanism I, one has to define a one-to-one association between an application program and a score that indicates the level of system knowledge required to use this particular application. The greater the knowledge required, the greater the score. Thus, applications could be classified in three broad categories: Applications requiring advanced knowledge of the system (system masters) scoring a total of 3 points, applications that indicate advanced knowledge of the system that worth 1.5 point and finally applications that require the absolute minimum level of sophistication for 0.75 points. Then, the arithmetic average of all the sampled application scores of a particular user could serve as a suitable quantification mechanism for this ITQA.

Hence, if F_{appscore} indicates a function designed to gauge the level of sophistication for a particular user in terms of the type of applications she invokes, then:

$$F_{\text{appscore}} = \text{Score}_{\text{app1}} + \text{Score}_{\text{app2}} + \text{Score}_{\text{app3}} + \dots + \text{Score}_{\text{appn}} / n \quad (6)$$

where n=number of recorded used applications for a user

Figure 6.6: Associating the executed application with a sophistication score

In order to satisfy the requirements of mechanism II, a set of application monitoring criteria has to be devised, in order to associate the usage of a particular application to a user sophistication level. This is a non-trivial task to achieve for a number of different reasons that are outlined below.

<p>Novice user:</p> <pre>56 ls 57 cat myfile.txt 58 vi myfile.txt 59 vi myfile2.txt 60 ls myfile2.txt 61 cd records/ 62 ls 63 cat myfile2.txt</pre>	<p>Experienced UNIX User:</p> <pre>131 smbstatus grep moamar* grep -I eudora 132 ls -lta showlogs*.log.gz 133 gunzip showlogs01*.log.gz 134 grep -i ^Nov showlogs01*.log grep -v authorized>novemberhits.log</pre>
---	---

Figure 6.7: Shell command invocation between novices and experienced users

For command-line based applications (such as UNIX shell or Windows Command Prompt programs), the way an application is started can be used to provide an indication of the level of knowledge of a

legitimate user. Figure 6.7 displays two sets of UNIX shell commands. The commands were drawn from excerpts of the UNIX command line history file of two users. The first one shows commands that originate from a novice UNIX user, whereas the second one originates from the shell history file of an experienced UNIX user.

The excerpts shown in Figure 6.7 indicate some fundamental differences in the application invocation process between experienced and novice UNIX users. Apart from the difference in the command vocabulary size discussed on earlier paragraphs, it is evident that advanced users tend to call applications with a larger number of arguments and options. For example, when considering the usage of common commands such as “cat” and “ls”, novice users tend to use “cat” as a command to list the contents of files. However, experienced users tend to employ it more along the lines of its original inception which is about concatenating file contents. The end result is an increase in the number of arguments passed on the command line.

Similar conclusions could be deducted for the number of command line optional flags, in order to modify the default behavior of the command. In the above excerpt, the experienced user invokes the “ls” command with additional arguments, in order to sort certain files according to their creation date. In contrast, novice users employ the command to just list the contents of their working directory, which is the default behavior.

Another important observation that distinguishes advanced users from novices is the employment of extensive I/O redirection features. The shell history command list for experienced users indicated approximately three times more frequent usage of command line pipes and other I/O output redirection features than the equivalent one for novice users.

Based on the previously made observations, command sophistication signatures could be devised, so that a certain sophistication score could be allocated for every invocation of the command. However, this technique has a certain number of disadvantages. An obvious one is concerned with the fact that the list of system commands as well as their respective argument options can grow easily. Producing (and also maintaining) a number of signatures for every command would be a non trivial task. The

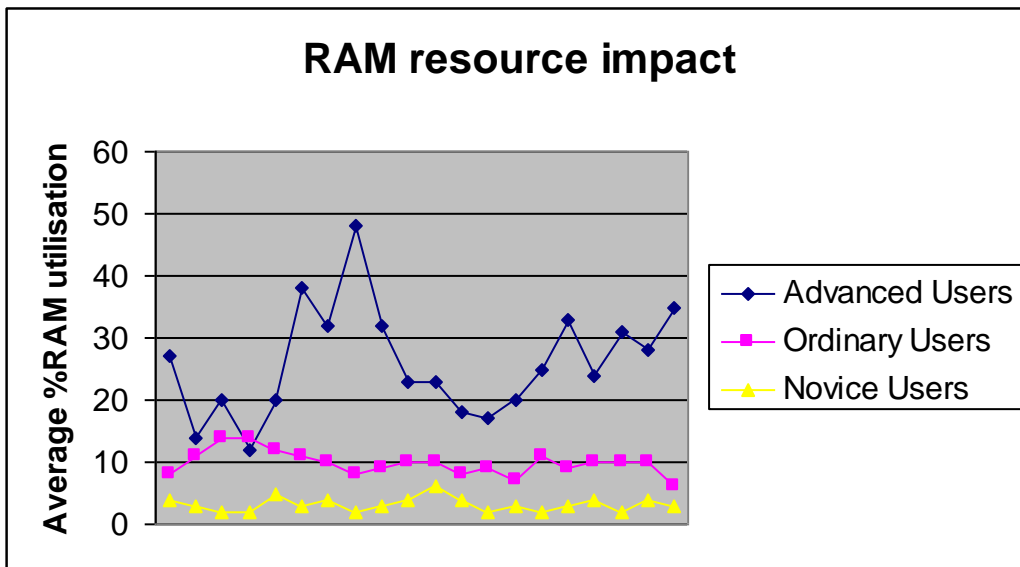
level of difficulty is also enhanced due to the fact that there are substantial differences in the command amongst different Operating Systems, or even amongst different hosts that run the same type of Operating System.

However, the most important drawback is concerned with the fact that the technique would not be useful for Graphical User Interface applications. Most modern Operating Systems such as Windows 2000/XP, MAC OS X, as well as UNIX/LINUX desktop systems use primarily graphical applications that are executed in standard ways, without leaving enough data for user sophistication classification. The profiling of user actions in a Graphical Environment would require some element of software re-engineering of the application in order to mark the events that can be used as sophistication metrics

Although the thesis is not concerned with pioneering IDS computational efficiency mechanisms or application-level monitoring frameworks, it is important to produce a pilot Insider Threat Prediction Model that could be used easily in a live system. Thus, for all the previously mentioned reasons, the employment of command line argument parsing and system call tracing is considered computationally expensive for the purposes of gauging user sophistication.

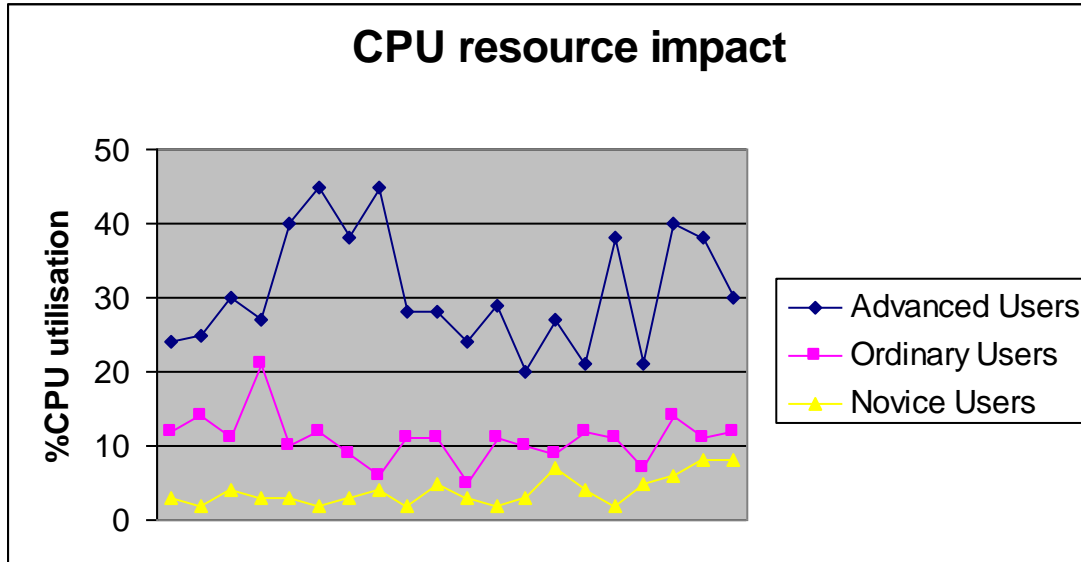
In an attempt to discover alternative metrics of user sophistication, the following paragraphs discuss the implementation details and the detailed numeric results of a survey that monitored 60 UNIX users in the National EMBnet Node, a scientific center located at the University of Oslo in Norway. The sample contained three categories of users that were pre-classified in terms of their documented professional role. Hence, the sample included:

- Advanced users: Includes system administrators and scientific personnel with substantial programming knowledge (software engineers, computer science and bioinformatics academic personnel) that have been users of the system for more than two years.
- Ordinary users: Scientists that had been using the server facilities for a minimum of 12 and a maximum of 24 months.
- Novices: Students who have recently attended an introductory course for using the UNIX system or users that have been using the system for less than twelve months.



	Advanced	Ordinary	Novice
Arithmetic mean	26	9.85	3.25
Standard Deviation σ	8,813864676	2,033275812	1,118033989

Figure 6.8: Average percentage of RAM utilisation



	Advanced	Ordinary	Novice
Arithmetic mean	30,9	10,95	3,95
Standard Deviation σ	7,98617226	3,316228041	1,959457497

Figure 6.9: Average percentage of CPU utilisation

The participants employed a series of generic applications, such as email and word processing programs, as well as specialized bioinformatics utilities such as the EMBOSS application suite [87], BLAST [88] and a variety of programming language interpreters and compilers.

There were an equal number of participants from all categories. A number of different metrics was employed, in order to verify both the breadth and depth of system users. All the results were collected by examining Operating System shell-level commands, as well as system resource utilization metrics obtained by standard Operating System utilities. A total of 20 ‘sessions’ per user were employed to collect the amount of data. In this Chapter, the term ‘user session’ refers to **all the commands and system resource impact indicators collected from the moment a user logs in until the time he logs out. This includes data from multiple user shell sessions.** The numbers were then averaged, in order to make certain conclusions about the different user categories.

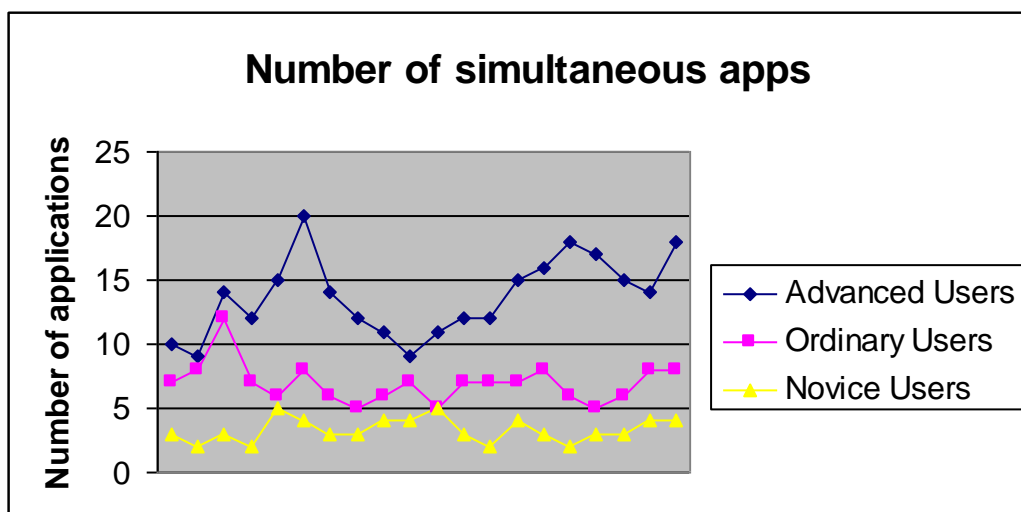


Figure 6.10: Average number of simultaneous applications per user session

The first important conclusion was that the level of user sophistication was proportional to the CPU and RAM utilisation. Advanced users on average consumed approximately three times more CPU and RAM than ordinary users. Advanced users also appeared to consume on average approximately ten

times more of these resources than the novice users. Figures 6.8 and 6.9 illustrate the distribution of values for these two metrics for all user categories.

The same conclusions could be deducted by looking into the number of applications used simultaneously (per user session) for the three user categories. In particular, the most sophisticated users employed on average twice as many simultaneous applications as the ordinary users and four times the average amount of simultaneous applications of novice users. Figure 6.10 summarizes these findings.

Combining all three of these metrics rather than using one of them is an essential action, in order to increase the reliability of the user sophistication gauging. This particular experiment provided mostly clear borders of distinction for all three user categories. However, in figures 6.8 and 6.10, small undesirable overlaps amongst different user categories can be observed. For instance, the RAM resource impact graph indicates an overlap between the Advanced and the Ordinary users category. The fourth sampled Advanced User value is well below the fourth Ordinary Users metric value. A greater degree of overlap can be also observed in Figure 6.10, between the Advanced and Ordinary Users as well as the Ordinary and Novice user groups.

A different applications environment could potentially increase the overlap of these metrics amongst the different user categories, making the process of user classification difficult. Consequently, even if we combine the CPU, RAM and number of simultaneous application metrics, we could not use their recorded maximum and minimum values. In an attempt to prevent this classification overlap problems, all three of these metrics are considered in the $F_{resutil}$ function, a mechanism that gauges user sophistication using these metrics, based on the recorded arithmetic average of these metrics. Then, F_{depth} would have the following form:

$$F_{depth} = F_{appscore} + F_{resutil} \Rightarrow$$

$$F_{depth} = (\text{Score}_{app1} + \text{Score}_{app2} + \text{Score}_{app3} + \dots + \text{Score}_{appn} / n) + S_{CPU} + S_{RAM} + S_{SIMAPPS}$$

(7)

Figure 6.11: The assessment of the legitimate user's depth of knowledge

S_{CPU} , S_{RAM} and $S_{SIMAPPS}$ represent the scores allocated for the measured CPU RAM and simultaneous applications metrics. However, prior assigning the raw values for S_{CPU} , S_{RAM} and $S_{SIMAPPS}$ in equation (7), a refinement of these metrics is also necessary, in order to smooth out the derived values and aid the process of eliminating the aforementioned overlaps. For each of these variables, if μ represents the arithmetic average of each metric for every user category, and x the recorded value of a metric per user, Figure 6.12 displays a set of equations that perform the necessary refinement.

$$\begin{aligned}
 & \mathbf{S_{CPU} = 1, \text{ if } (\mu_{\text{ordinary}} < x \leq \mu_{\text{advanced}}) \text{ OR } (\mu_{\text{ordinary}} < x \text{ AND } x \geq \mu_{\text{advanced}})} \\
 & \quad \mathbf{S_{CPU} = 0.5, \text{ if } \mu_{\text{novice}} < x \leq \mu_{\text{ordinary}}} \\
 & \quad \mathbf{S_{CPU} = 0.1, \text{ if } 0 < x \leq \mu_{\text{novice}}} \\
 & \quad \mathbf{(8)} \\
 & \mathbf{S_{RAM} = 1, \text{ if } (\mu_{\text{ordinary}} < x \leq \mu_{\text{advanced}}) \text{ OR } (\mu_{\text{ordinary}} < x \text{ AND } x \geq \mu_{\text{advanced}})} \\
 & \quad \mathbf{S_{RAM} = 0.5, \text{ if } \mu_{\text{novice}} < x \leq \mu_{\text{ordinary}}} \\
 & \quad \mathbf{S_{RAM} = 0.1, \text{ if } 0 < x \leq \mu_{\text{novice}}} \\
 & \quad \mathbf{(9)} \\
 & \mathbf{S_{SIMAPPS} = 1, \text{ if } (\mu_{\text{ordinary}} < x \leq \mu_{\text{advanced}}) \text{ OR } (\mu_{\text{ordinary}} < x \text{ AND } x \geq \mu_{\text{advanced}})} \\
 & \quad \mathbf{S_{SIMAPPS} = 0.5, \text{ if } \mu_{\text{novice}} < x \leq \mu_{\text{ordinary}}} \\
 & \quad \mathbf{S_{SIMAPPS} = 0.1, \text{ if } 0 < x \leq \mu_{\text{novice}}} \\
 & \quad \mathbf{(10)}
 \end{aligned}$$

Figure 6.12: Refinement formulae for S_{CPU} , S_{RAM} and $S_{SIMAPPS}$

The $F_{\text{sophistication}}$ values derived by the equations (7)-(10) are plotted in Figure 6.13. The graph indicates no overlap amongst the three user categories. One can now observe more clearly the borders of distinction amongst the different user categories. Hence, it is assumed that the combined application and refinement of metrics in this manner improves the reliability of the user sophistication gauging process. In this particular experiment, the results indicated that advanced users achieve an $F_{\text{sophistication}}$ score that ranges from 10 to 11.9 units, ordinary users are placed in the range of 5.4 to 9.7 and novice users scores were measured in the range of 1.4 to 4.7 points. The results indicated no instances of user misclassification.

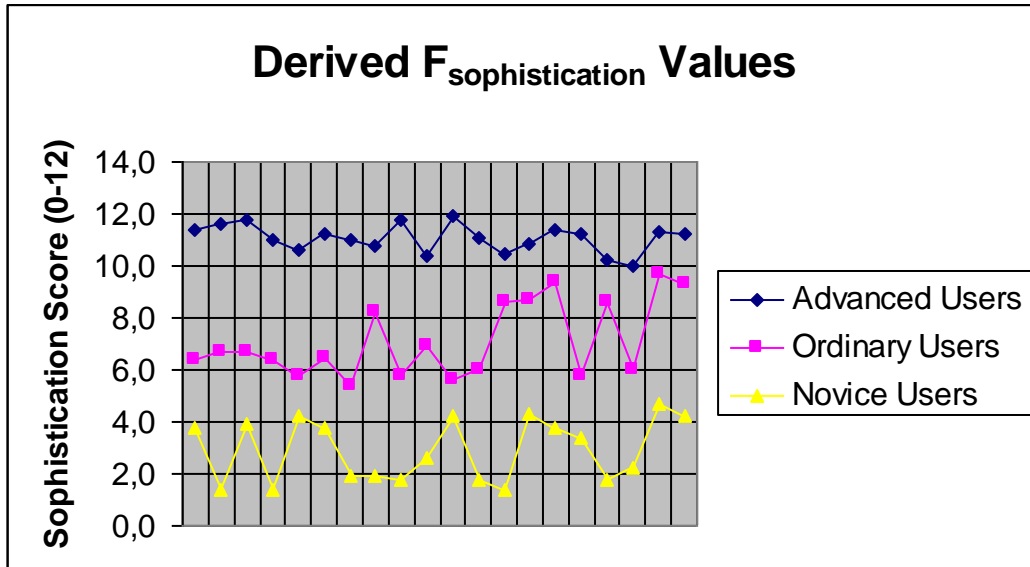


Figure 6.13: The distribution of $F_{sophistication}$ values

Consequently, a methodology could be derived that would allow an automated process to classify users in terms of their level of sophistication. The first step of this methodology can be achieved by selecting a user sample which contains an equal number of users from each sophistication level. The next step involves the process of training the model by measuring repeatedly the metrics for people of the same category, in order to establish minimum and maximum $F_{sophistication}$ values for each user sophistication level. These values can then be used for subsequent measurements of new users, in order to gauge their level of sophistication and they are specific to the number and type of applications of a particular computational environment.

If the initial sample user categorization according to the user's documented role and experience is false, the model will yield inaccurate results. Therefore, the entire procedure requires intervention from experts for the purposes of validating the training user sample. Moreover, because $F_{sophistication}$ refers to specific computational environments, if new applications are installed on the target system, the function will require re-sampling of the training values, in order to function correctly. These two limitations represent two important weaknesses of the method. These weaknesses increase substantially the setup administration overhead of the method in today's fast evolving IT infrastructures. Nevertheless, the $F_{sophistication}$ component function represents a novel experimental approach that did provide accurate classification results in the experiment.

This step concludes the qualification and quantification of metrics suitable to measure legitimate user sophistication.

6.3.2.2 Modeling user actions by monitoring file and network operations

The fifth Chapter of the thesis presented an Insider Misuse oriented taxonomy based on a number of system level detectable consequences. It was then argued that file and network level operations could be employed as a mechanism for revealing Insider IT misuse acts (Figure 5.5 and Figure 5.7). Based on these initial thoughts, it is possible to construct pattern matching signatures describing user file and network operations that would indicate the signs of forthcoming misuse incidents.

Driven mainly by the proposed Insider Misuse Prediction Taxonomy, a suitable file-level collection of ITQAs should include the following metrics:

- *Existence of file*: The ITPM system should be able to search files owned by a particular user that match one or more of the following criteria:
 - File metadata attributes: Recognised by file name, type of file (examples .exe, .mp3, hidden folder or system file) or special file attributes (whether the file is executable, read only, etc).
 - File contents: The files contain certain content that should be detected at various data type levels, depending on the file encoding format (ASCII, UTF versus binary format) and the type of information that the system attempts to intercept.
 - File size: Sometimes, it is easier to intercept file-level evidence if we know the exact size of the file. There are plenty of examples of malicious code or improper content that contain files of certain type and constant size.
- *Access of file*: The fact that a particular user accesses a specific file in certain ways should also serve as an ITQA in the ITPM system:
 - File access time: When a particular file is accessed might be important
 - Access mode: Whether the file is accessed in real-only or read-write mode.
- *User area space consumption*: Previous chapters elaborated on aspects of accidental misuse related to over-utilisation of system resources. It should be possible to compare the user overall space consumption in relation to:

- A pre-set limit set in the system (often referred to as user disk quota).
- The overall storage capacity of the disk where the user area resides.
- The disk consumption of the previous user session, so that the rate of increase of disk space consumption can be established. Moreover, if *currutil* represents the disk utilisation at the end of the current user session and *prevutil* the disk utilisation at the end of the previous user section, the ratio *currutil/prevutil* represents the required metric. The calculated ratio should then be compared against a threshold ratio value to indicate whether the user disk consumption is growing alarmingly fast.
- The existence of certain file types, so for instance it is possible to see if most of the hard disk space is occupied by file types that are unauthorised.

It should be noted that the ‘existence of file’ as well as the ‘access of file’ attributes hold true for directories. In fact, in most widely employed Operating Systems, directories are special files that act as containers for holding files [72, 73].

Following a similar line of thinking for network operations monitoring, the following set of ITQA metrics should be available to the ITPM system.

- Existence of user related network connection on the host: The ITPM system should be able to evaluate potential threats from one or more the following network endpoint attributes:
 - Source IP address: Employed to evaluate inbound host network connections associated with a particular user.
 - Destination IP Address
 - Transport Protocol employed (UDP versus TCP)
 - Source and destination port number employed.
 - Application Payload contents match (*optional facility*): The ability to match a series of bytes located in the payload area of the Protocol Data Unit (Figure 2.6). This is designated as an optional attribute mainly due to the performance impact it might have on a busy computational environment. Computer microprocessors are getting faster but technologies such as Gigabit Ethernet and the ever increasing

computational demands of modern desktops create serious performance overheads in network-based intrusion detection systems [19]. Consequently, having an attribute that requires interception of network data at great speeds is not a good idea in terms of efficiency and reliability.

- Network resources consumption: A series of ITQA metrics employed to address the extent of the likelihood of a legitimate user over-utilizing the networking subsystem of a host.
 - Number of connections employed: The number of host network connections associated with the user in relation to a threshold value. The establishment of a network connection requires the allocation of computational resources such as special kernel data structures [72,73]. There are a finite number of these data structures and their exhaustion could seriously affect the proper operation of a system. Consequently, the threshold value should always be a fraction of the total number of network endpoints that an Operating System can allocate.
 - Total number of Mbytes exchanged (sent and received) in relation to a threshold quota value (again in Mbytes) for all network connection associated with the user during a single user session.
 - Number of Mbytes exchanged per user connection (sent and received), in relation to a threshold quota value (in Mbytes) during a user session.
 - Rate of increase of the Total number of Mbytes exchanged (sent and received) in relation to the total number of Mbytes of the previous section.

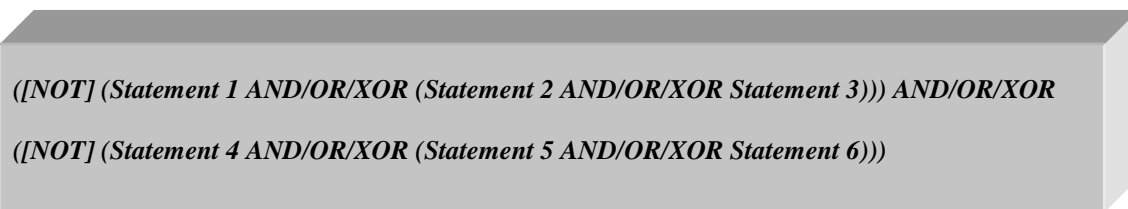
The reader should note that the proposed network-connection specification addresses the Transmission Control Protocol/Internet Protocol (TCP/IP) [89] network system implementations. There are computing systems that run different network protocols such as bespoke implementations of the Open Systems Interconnect (OSI) [90] or the Systems Network Architecture (SNA) [91] family of protocols. However, these protocols are associated exclusively with older computer equipment and they are rarely encountered in new computing installations. TCP/IP is now the ubiquitous standard. As a result, the proposed ITPM network-level operations specification has ignored OSI, SNA and other more proprietary networking architectures.

6.3.2.2.1 File and network operation specification statements

After drawing the file-level ITQA list, the next step is to combine them together, in order to describe a range of particular file operation scenarios. This can be achieved by means of suitably encoded statements. The statements could act as the basis of a system-orientated signature construction mechanism that produces separate signatures for file and network operations. The constructed signature could then act as a pattern matching mechanism intended to intercept (in combination with other indicators) insider threats at file system and network level. The syntax and the rules of combining these statements are presented on the following paragraphs.

The general encoding format contains a statement file operation indicator such as 'existsf ' and a series of ITQA attributes that identify further the object(s) of the operation. The ITQA attributes are separated by colons (:) and square brackets indicate optional parts of the statement. Curly braces indicate optional nesting of statements, something which is discussed in latter paragraphs. Finally, whenever a single wildcard (*) character is assigned to an ITQA attribute, it forces the statement to be considered against all the possible attribute values.

The statements can be combined together using four logical operators and control blocks indicated by parentheses as illustrated in Fig. below. This schema gives the signature mechanism the ability to define complex file and network-level events. The NOT operator negates the meaning of the entire parentheses block, whereas the AND/OR/XOR operators are used to define combinations of statement conditions according to their boolean algebra meaning [92] (Figure 6.14).



*([NOT] (Statement 1 AND/OR/XOR (Statement 2 AND/OR/XOR Statement 3))) AND/OR/XOR
([NOT] (Statement 4 AND/OR/XOR (Statement 5 AND/OR/XOR Statement 6)))*

Figure 6.14: The structure of file/ network operation specification statement

This logical operator scheme can also be employed in the definition of indicated ITQA attributes, in order to enhance further the expressiveness of the statements. Hence, the generic structure of a discrete file/network-level signature is presented below:

([NOT]((statementoperator1:<[NOT]((attribute1value1[OR/AND/XOR/]attribute1value2>):<attribute2>]...{nestedstatementoperator1:...})[OR/AND/XOR](statementoperator2:<attribute1value2>....)))

Type of statement	Statement Syntax	Example
<i>Existence of files</i>	existsf:<filename>:[<filetype>]:[<readflag>]:<writeflag>:<execflag>:<suidflag>]:[<contents>]:[<size>]:[<nooffiles>]:[<location1, location2,...>]	existsf:<*mutella*>:<tar OR tar.gz>:<s>:<u>:<s>:<u>:<\$HOME,usr/src> Explanation: Find a user owned file of type tar or tar.gz that contains the string mutella as part of its name, has the read and execute flags set for the user and if it is not under the user's home directory, it might also be under the user home or /usr/src directories.
<i>Existence of directories</i>	existsd:<dirname>:[<readflag>:<writeflag>]:<execflag>]:{existsf:<filename>...:<size>[OR][AND][XOR]existsf:<filename>...}:[<location1, location2,...>]	existsd:<*mutella*>:<s>:<s>:<s>:{existsf:<mutella>:<binary> AND existsf:<AUTHORS>:<asciitext>:<contains:Mutella Project>} Explanation: Find a directory with the approximate name mutella, that has read, write and execute permissions for the user. This directory should also contain (amongst other) two files: one binary called "mutella" and one ascii text called "AUTHORS" which has the string "Mutella Project" in its contents.

Table 6.2: File-level existence statements

In order to further illustrate the previous complex expression starting with file-level operations, Table 6.2 summarises the syntax for encoding file operation existence statements. The 'existence ITQA' operation indicators ('existsf' and 'existsd') are concerned with the presence of certain files and

directories in the user's home area or elsewhere. The <filename> and <dirname> attributes specify the name of the entities in question. It should be noted that when the name of the files cannot or need not to be precise, these attributes can act as regular expressions by inserting wildcard characters at the beginning and end of the <filename>/<dirname> string values. This allows the statement to act on a variety of potentially threatening files and directories.

This is also true for the <contents> attribute that aims to match certain file payloads that could be deemed as indicators of forthcoming misuse threat. The attributes (<readflag>, <writeflag>, <execflag>) are optional components that relate to what permissions the user has in relation to the entity he is trying to access. Their purpose is to refine the file entity selection criteria as much as possible.

The <size> and <nooffiles> parameters of the 'existsf' statement are integers. The earliest ITQA states the size of the file or directory entity in megabytes, whereas the latter <nooffiles> defines a threshold value of files that meet the criteria of the 'existsf' statement. Lastly, the search for files that satisfy the ITQA criteria is performed by default on the user's home area. However, some of the files in question might be located outside the user home area. In these cases, the <location1, location2,...> optional attribute might be employed, in order to define a number of locations (absolute path), where the search can also take place.

Table 6.3 displays the syntax for the '*accessf*' and '*accessd*' statements. They address the current list of files and directories the user is trying to access. The <accesshourrange> attribute is an optional component that indicates if files are accessed within a suspicious range of hours, and hence addressing specifically malicious alteration or theft of information.

Type of statement	Statement Syntax	Example
<i>Access of file</i>	accessf:<filename>:[<accesshourrange>] :{existsf: <filename>...:<location1, location2,...>}>[OR] [AND] [XOR] existsf:<filename>...:<location1, location2,...>}}	accessf:<*.doc>:<04-06>:{existsf:<*>:<doc>:</storage/p rototypes>} Explanation: match all files being accessed between the hours of 4 and 6 in the morning that are of type *.doc and reside under the /storage/prototypes directory.
<i>Access of directory</i>	accessd:<dirname>:[<accesshourrange>]]:{existsf:<filename>...<location1, location2,...>}}	accessd:</storage/prototypes>:<04-06>:{existsf:<*>:<doc>} Explanation: Match any access to the directory /storage/prototypes that contains one or more word documents, between the hours of 4 and 6 in the morning.

Table 6.3:File and Directory access statements

The ‘Filesystem over-utilisation’ statements are designed to handle the problem of disk space over-utilisation, as shown on Table 6.4. The ‘checkrelq’ statement examines how much of the disk space quota the user has utilised. <percent-util> is an integer (1-100) that expresses a disk space per-cent consumption threshold in relation to a pre-defined limit in Mbytes (<quota>). When the user’s disk consumption exceeds that threshold, the statement becomes true and hence a part of the event that is described by the file-level signature produces a match. In contrast, when a user quota is not defined, checks could be made against the entire disk partition where the user area resides (checkdiskq statement). In that case, <dlimit> expresses the size of the disk/partition in Mbytes. This would be useful to address situations such as the one described in Appendix D of the thesis, when disk quotas are not enforced on computer systems.

Type of statement	Statement Syntax	Example
<i>User relative quota check</i>	checkrelq:<percentutil>:<quota>	checkrelq:<75>:<2048> Explanation: Has the user consumed three quarters or more of his 2 Gigabyte quota?
<i>Disk quota check</i>	checkdiskq:<percentutil>:<dlimit>	checkdiskq:<50>:<40960> Explanation: Has the user exceeded half of the 40 Gbyte disk/partition?
<i>Disk consumption per file type check</i>	checkfiletypeq:<filetype>:<percentutil>:<quota>	checkfiletypeq:<mpeg OR avi OR wmv OR mpg OR mp3>:<50>:<2048> Explanation: Has the user exceeded half or more of his 2Gbyte quota on multimedia files?
<i>Rate of disk consumption increase check</i>	Checkdiffq:<threshold_file_quota_ratio>:<quota>	Checkdiffq:20:<2048> Explanation: Did the user quota consumption utilisation grow by 20% or more in relation to the previous user user quota utilisation, when the quota is set to 2Gigabytes?

Table 6.4: Filesystem over-utilisation statements

The ‘checkfiletypeq’ statement is more specific and it examines disk utilisation associated to particular file types. This type of statement would be useful in specifying threats indicated by large disk consumption figures associated with particular file types. A good example of this situation would be a consistent growth of user disk space consumption associated to mp3, mpeg, avi or other multimedia file formats. When this information is combined with additional indicators (such as the presence of certain running processes) could for example indicate the presence of unauthorised file sharing software.

The last filesystem over-utilisation statement ‘checkdiffq’ examines the rate of overall disk space consumption per user. <threshold_file_quota_ratio> expresses an indicated ratio of the current

session's quota utilisation over the previous session quota utilisation, in order to indicate whether the user's disk storage requirements are growing too fast.

Earlier paragraphs mentioned the nesting of statements inside others. This feature is desirable for combining the file operation statements into composite expressions that increase the accuracy of the file operation signature specification. For instance, an 'existsf' statement may nest inside an 'existsd' directive, in order to specify additional criteria about the files of a particular directory and hence increase the specification capabilities of the 'existsd' operator. This is also true for the nesting of 'existsf' statements inside 'accessf', 'accessd' and 'checkfiletypeq' directives.

Similar encoding considerations should be taken into account for the construction of network-level signatures. Table 6.5 provides an overview of the required network operations statements.

Type of statement	Statement syntax	Examples
Existence statement for network connection	existsnet:<destination_ip/FQDN>:<source_ip/FQDN>:<transport_protocol>:<source_transport_port>:<destination_transport_port>:[<payload_match>]	existsnet:<129.240.4.5>:<www.warez.com>:<tcp>:< > 1025>:<80> Explanation: Has the user visited the www.warez.com website.

<p>Network Quota statements</p>	<p>checknetendpointq:<threshold _number_of_endpoints>:{exist snet:<destination_ipORFQDN >...}}</p> <p>checknetbytesessionq:<percen tutil>:<net_quota></p> <p>Checknetdiffq:<threshold_net _quota_ratio>:<net_quota></p>	<p>Checknetendpointq:<30>:{existsnet:< 129.240.4.5>:<ftp.freemp3.org>: <tcp>:< > 1025>:<21>}</p> <p>Explanation: has the user launched 30 or more connections to the ftp.freemp.org server?</p> <p>Checknetbytesessionq:<80>:<1024></p> <p>Explanation: Has the user utilised more 80 per cent or more of his 1 Gbytes network quota in the current session?</p> <p>Checknetdiff:<30>:<1024></p> <p>Explanation: Did the user relative net consumption (of a 1024 Mbytes net quota) grew more than 30% in relation to the previous user session?</p>
---------------------------------	--	--

Table 6.5: Network operation statements

The ‘existsnet’ statement facilitates the network connection orientated threat detection. The destination and source Internet Protocol Addresses can be defined in their numeric or Fully Qualified Domain Name (FQDN) format (<destination_ipORFQDN> and <source_ipORFQDN>). The <transport_protocol> and <transport_port> identifiers dictate the transport protocol employed (tcp or udp) and the port number (integer).

The <transport_port> identifier can specify a specific port (integer) or range of ports in a number of different ways, as shown on table 6.6 below. The last optional ITQA <payload_match> searches for an ASCII-based keyword in the payload (data) area of the packet.

Type of source/destination transport_port specifiers	Syntactic example	Meaning
Single port specification	<i>80</i>	Match the criteria for connections for/from port 80 only.
Range of ports	<i>30000-40000</i>	Match the criteria for any port in the range 30000-40000.
<i>Greater than or less than</i>	<i>i) > 1024</i> <i>ii) < 1024</i>	i)Match all ports greater than 1024. ii)Match all ports that have a value less than 1024.
Composite with logical operators	<i>80 XOR (120-150 OR 3600-6000)</i>	Match port 80 or exclusively a port that belongs to the 120-150 or 3600-6000 range not both.

Table 6.6: Transport port attribute values

The Network Quota Statements (Table 6.5) function in a way that is similar to the filesystem over-utilisation statements. However, instead of dealing with bytes stored on a medium, the volume of incoming and outgoing byte sequences is considered. ‘checknetendpointq’ matches either a total number of endpoint connections (<threshold_number_of_endpoints> is an integer) or a number of endpoint connections that meet certain criteria. The latter optional functionality can be achieved by nesting an ‘existsnet’ directive inside the ‘checknetendpointq’ statement, in order to focus its scope on specific connections.

The ‘checknetbytesessionq’ statement is equivalent to the ‘checkrelq’ one. The <net_quota> attribute defines the maximum total number of Mbytes (received and sent) per user session and the statement becomes true if the current relative utilisation is equal or greater than the number quoted by <percentutil>. Finally, ‘checknetdiffq’ evaluates the growth of the relative network quota utilisation (<threshold_net_quota_ratio>) in relation to the previous user session.

This concludes the presentation and justification of the file and network operation specification statements. In order to demonstrate the established specification schemes and apart from the

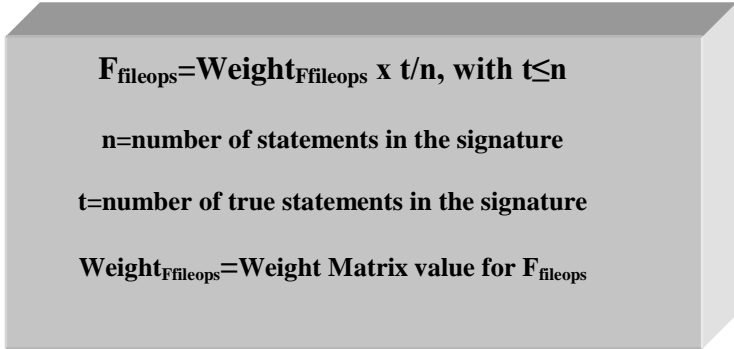
specification table examples, a step-by-step description of constructing a sample misuse prediction signature is provided in section 6.4 of this chapter. The next subsection explains how to produce a threat prediction value from a file/network specification signature.

6.3.2.2.2 Derivation of the Ffileops and Fnetops formulae

After the definition of the file and network statements and their respective syntactic rules, the Ffileops and Fnetops formulae that calculate the behavioural threat components must be constructed, with regards to equation (3) of section 6.3.2. In order to derive these formulae, one has to consider the structure of file/network operations statement, as presented in figure 6.14. The figure shows the structure of a single file or net operation statement. A collection of these statements statements that *are all necessary* to verify the presence of a forthcoming threat constitutes a specification signature:

$$S_{file/net} = \text{FileStatement1}, \text{FileStatement2}, \text{FileStatement3}, \dots, \text{FStatementn}$$

As the ITPM system evaluates the statements, some of them are going to be true (find conditions that match their attributes) whereas others are going to be false (not satisfy their attributes). Hence, if there are t true statements in the signature, then the formula of $F_{fileops}$ is defined in figure 6.15 and in the same way figure 6.16 displays the F_{netops} equation. Both functions are defined as the ratio of the number of their true statements over the total number of statements of their signature multiplied by their Weight Matrix value (Table 6.1).



$$F_{fileops} = \text{Weight}_{F_{fileops}} \times t/n, \text{ with } t \leq n$$

$$n = \text{number of statements in the signature}$$

$$t = \text{number of true statements in the signature}$$

$$\text{Weight}_{F_{fileops}} = \text{Weight Matrix value for } F_{fileops}$$

Figure 6.15: F_{fileops} formula



$$F_{netops} = \text{Weight}_{F_{netops}} \times t/n, \text{ with } t \leq n$$

n=number of statements in the signature

t=number of true statements in the signature

Weight_{F_{netops}}=Weight Matrix value for F_{netops}

Figure 6.16: F_{netops} formula

This constitutes a simple pattern matching mechanism, whose reliability relies essentially on the definition of the signature. If the decomposition of the file/network threat components as signature statements is not accurate, then the respective functions are going to be inaccurate, yielding false positive or negative alarms.

6.3.2.3 Modeling legitimate user actions by command line signature processing

Whilst earlier discussions presented mechanisms of associating misuse threat levels to legitimate user sophistication as well as network and file-level actions, this section will focus on the execution order of legitimate user actions. This is an equally important aspect of the insider threat estimation process.

Section 6.2 discussed the Schultz framework's [83] proposed metrics of "deliberate markers" and "preparatory behaviour". Apart from the knowledge of user actions (file access and program execution), these metrics imply the notion of a process, in the sense of an **ordered collection of user actions**. Hence, one could encode a user session as an ordered sequence of commands, together with their respective arguments. Figure 6.17 below illustrates this concept.

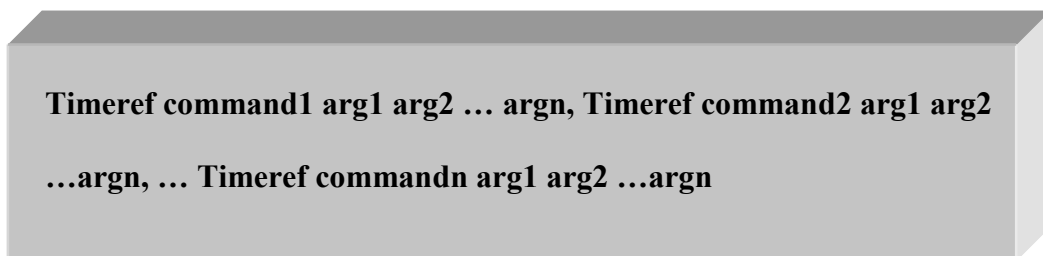


Figure 6.17: Ordered Shell command sequence

The 'Timeref' is a timing indicator that records when a command was issued. This is useful when someone wishes to correlate data (considering user data from multiple hosts) or establish whether the

commands were executed by an automated script or manually by the user itself. The commandx indicators reveal the users program execution, whereas argvx might reveal further details about the commands and the associated file access operations.

Each of the recorded user action sequences could then be compared against a set of sequences that contain commands of known misuse actions. The way the comparison is done is a key design feature of an Insider Threat Prediction Model. Whilst latter paragraphs will discuss the details of the comparison scheme, it is useful to emphasize that the problem of predicting legitimate user misuse is, in essence, a sub-domain of sequence prediction analysis. It should be acknowledged that the idea of analysing user command sequences for the purposes of predicting user actions is not a new one.

Lee [93] and Greenberg [94] have collected and analysed UNIX command line data and tried to discover simple patterns of repetition in various command sets. Their results indicate that the likelihood of repeating certain commands is quite high. However, they both encountered several problems as individual usage patterns varied substantially, producing undesired effects in their data sets and reducing substantially their predictive reliability.

In their attempt to enhance the adaptation of a user interface to an individual's pattern of use, Davison and Hirsh [95] devised the 'Incremental Probabilistic Action Modelling' (IPAM) algorithm. Their algorithmic approach represents a Machine Learning method that involves the analysis of UNIX command line data sets for the purposes of calculating the probability of future commands. The authors employ the assumption that each command in a sequence depends on the previous one. Then, an entire user command sequence could be modelled by counting the number of times each command $x+1$ followed a command x and that could determine the likelihood of going from one command to another.

Davison and Hirsh experiments included command line histories from a sample of 77 users. A total of approximately 168,000 commands were analysed (on average 2000 commands per user), in order to evaluate the effectiveness of the IOLA method. Since the number of sampled commands was not the same across all users, it was necessary to include two types of performance indicators. 'Macroaverage' results considered the predictive accuracy of each individual user and then average the figure over all

users. On the contrary, ‘microaverage’ results took into consideration the number of correct predictions made across the entire user base divided by the total number of commands for all users. Based on these assumptions, the authors reported a 39.9% macroaverage predictive accuracy and a respective 38.5% of microaverage predictive accuracy.

Although the IPAM algorithm represents a promising approach to aid the construction of efficient User Interfaces, it is not useful for the purposes of Insider Threat Estimation. The goal of Davison and Hirsh was to devise a method to predict generic user command sequences, without considering Domain Specific information. Legitimate misuse incidents require the representation of domain specific knowledge. In addition, the IPAM algorithm ignores command line arguments, as it considers only the execution of commands. These arguments would indicate the potential target of a misuse act, an essential element of an insider threat estimation process.

In addition, the computational efficiency of the model could also be a concern. Davison and Hirsh have proved the efficiency of the IPAM algorithm is greater in relation to other relevant machine learning approaches based on statistical computations. Chapter 2 elaborated on the disadvantages of anomaly detection in terms of computational efficiency. Misuse detection methods based on faster pattern matching approaches could potentially offer a much more efficient approach in a threat estimation model.

Based on these assumptions, the proposed modelling approach would opt for an efficient pattern matching method that would take into consideration domain specific aspects and would include command line argument data. Figure 6.17 illustrated the proposed data schema for representing the sequence of user actions. Let δ_m denote a sequence representing m sampled commands (together with their respective arguments) of a legitimate user and v_n represent a misuse signature (command sequence with respective arguments) of length n that describes a known misuse act. Assuming that sequence u_n is complete (it contains all the steps required to commit a particular insider misuse act), the similarity between these two sequences can serve as an indicator of insider threat. The more similar the sequences, the greater the likelihood that δ_m represents an indication of forthcoming misuse activity.

This raises the question of how the similarity between these two sequences should be evaluated. After sampling the user commands and encoding them in the form described in Figure 6.17, it is also necessary to associate each system command to an integer. This command enumeration step is necessary, in order to produce a more computationally efficient sequence format. For example, if the UNIX command ‘/sbin/tcpdump’ is represented as an integer ‘1200’, it will use less storage space and will require less CPU cycles in a pattern matching algorithm than the full character string format. Although the differences might not seem great, in large sequences that are repeated on hundreds or thousands of users, this refinement could have a substantial impact on busy systems. In contrast, the command arguments are not enumerated because it is not trivial to produce lists of all the possible encoded combinations for each of the system commands. Each command operates with operands that change all the time, as well as with options that can grow exponentially.

After these prerequisites, an algorithm can now be devised, that scores the level of similarity between a legitimate user command sequence and a pre-constructed misuse signature. The steps of such an algorithm are outlined below:

- i) The ‘timeref’ indicators are removed. They served as a way to preserve the order of commands when data are correlated from different sources, but they are no longer needed at this stage.
- ii) Each of the the δ_m and v_n sequence commands are encoded in the following string format:

CcommandcodeAargument1argument2...-##8#

C, ‘A’ and ‘##8#’ are standard sequence formatting strings. The numeric code of the command lies between the ‘C’ and the ‘A’ strings, whereas the command arguments are located between the ‘A’ and the command termination sequence string ‘##8#’. It should be noted that any white space characters (one or more space characters or tabs) are removed from the arguments, before they are placed in the proposed sequence format. Many commands of the UNIX and Windows command prompt interface are insensitive to white space. This can complicate the pattern matching process and hence white space is removed. The end result of this step is the creation of two strings: Slegitimate which represents the encoded δ_m and Ssignature which represents the encoded misuse signature.

- iii) The Slegitimate string is converted into an array of strings of size m (Alegitimate), where m is the number of the encoded commands in the sequence. Each element of the array represents a string that contains a single command of the sequence as encoded in step ii. The same procedure is repeated for the u_n misuse signature sequence, in order to produce an array of strings of size n (Asignature).
- iv) The similarity between these two arrays (and hence between the legitimate user and misuse signature sequences) is defined by the following procedure described below in pseudo-code notation:

```

outer_loop: for (i=0 i<=m i++) {
    if(sizeofAsignature!=0) {
        inner_loop:for (j=0 j<=n j++) {
            if(Alegitimate[i] == Asignature[j]) {
                number_of_matches++
                left shift Asignature by one element
            }
        }
        } else {
            return (100 * (number_of_matches/n))
        }
    }

```

Figure 6.18: Command sequence similarity search algorithm

In other words, the similarity between a legitimate user command sequence and a misuse signature can be expressed as the ratio of the number of commands from the legitimate user sequence that match the misuse signature commands over the total number of commands that describe the particular misuse signature. Figure 2.5 of Chapter 2 provided an illustration of the temporal basis of computer intrusion incidents. Insider misuse incidents follow an identical temporal pathway. Some initial actions pave the way for the next steps towards the misuse act, until the insider misuse incident is complete. The greater the number of steps matched in the order they were defined in the misuse signature, the more probable the eventuality of a particular incident.

From a computational perspective, this algorithm represents a standard element-by-element comparison of two arrays. An array is an ordered collection of data elements which represents one of the most widely available data structures, as it is employed by all popular high-level programming languages. The choice of the array data structure was made because it is important to perform an ordered comparison driven by the misuse signature command sequence. Kumar and Spafford [26] emphasize the existence of a strict (ordered) sequence of actions as a primary mechanism for producing a signature specification.

It is expected that the running time of this algorithm is proportional to the signature and user command array sizes. In formal terms, theoretical algorithmic running time estimations are quoted in “big-Oh” notation, as described by Brassard [96]. By convention, the growth rate of an algorithm’s running time is expressed as a function of its input size. Thus, if m and n are integers that represent the sizes of the Alegitimate and Asignature arrays respectively, it can be shown that the worst-case scenario running time of the proposed algorithm is proportional to the product of m and n , or that the algorithm is an $O(mn)$ one.

Every element of the Alegitimate array (outer_loop) is tested for equality against every element of the Asignature array. Every time a match is found, the matching Asignature element is disregarded and the total number of comparisons decreases, as the size of the Asignature array is reduced (left shift statement). Consequently, the worst case scenario is defined when no match can be found between the elements of the two arrays. Then, the total number of comparisons can be calculated as shown below:

$$\begin{aligned}
 \text{Number of comparisons} &= (m_0 \times n) + (m_1 \times n) + (m_2 \times n) + (m_3 \times n) + \dots + (m_{m-1} \times n) = \\
 &= n \times (m_0 + m_1 + m_2 + \dots + m_{m-1}) = \\
 &= n \times (m)
 \end{aligned}$$

Appendix E (section E3) provides an implementation of this algorithm (script realltimeomon.pl) using the PERL programming language. The script was then executed a number of times, accepting a signature of 4 commands ($m=4$) against various legitimate user command sequences of varying sizes.

The test sequences were carefully constructed so that no match could be found between their elements and hence simulate the worst case scenario conditions. Figure below illustrates the results of these runs on a Pentium III 1Ghz capable LINUX Workstation with 512 Mbytes of RAM. The results of the graphs below verify that the various aspects of the computational performance of the algorithm are proportional to the size of its input.

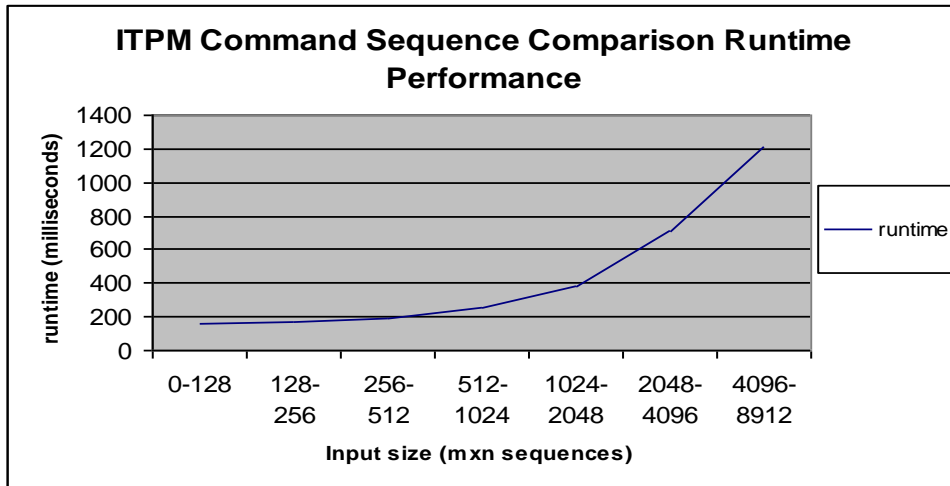


Figure 6.19: Runtime versus input size

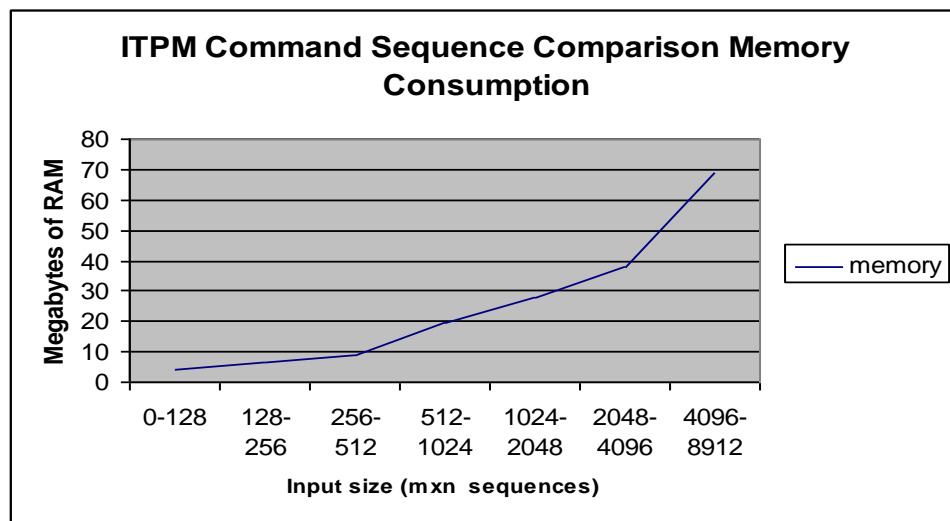


Figure 6.20: Memory footprint size versus input size

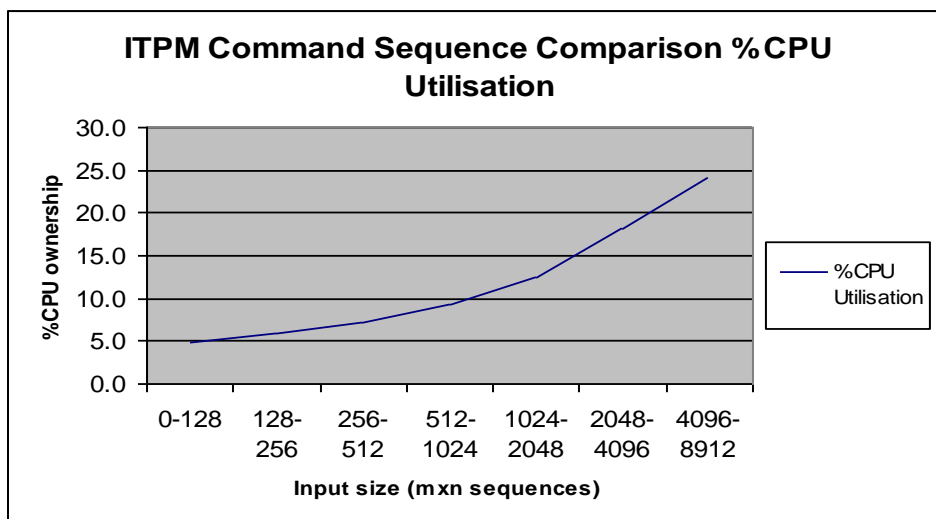


Figure 6.21: Relative CPU utilization versus input size

The memory and CPU requirements were also measured. Figure 6.20 indicates that if the mxn product addresses a number of sequences smaller than 1024, the proposed algorithm draws an amount of memory that could be easily served by a modern desktop computer (approximately 20 Mbytes were required when most modern desktops at the time of writing have at least 12 times that amount of memory in RAM).

However, the same conclusion cannot be derived about the relative CPU utilisation (Figure 6.21). The measurements were taken when 4 other processor intensive tasks were running (5 processes in execution). In the region of 512-1024 input sequences, the relative CPU consumption was just under 10% of the CPU time, for a running time of less than a quarter of a second. At a first glance, this does not appear to be a high CPU load, but if one considers the fact that the aforementioned utilisation figure is associated to a single signature comparison for a single user, a different perspective becomes apparent. Since, more than one misuse signatures will need to be examined against a single user command sequence, then a multi-user system would require even further additional computational resources (especially CPU time), in order to accommodate for the needs of the threat prediction process. Consequently, it becomes apparent that this algorithm requires further optimisations, in order to become operational in large production-grade systems. The details of such optimisations and the factors that affect the performance of the algorithm are discussed in detail in Chapter 7 of the thesis.

6.4 The production of a multi-level signature

The previous sections of this Chapter explained in detail the ITPM functions and they way they encode their data. They did not explain how all these mechanisms can be combined together to form a working model. The combination of the ITPM component functions is presented in the following paragraphs and will be accompanied by a discussion of how to apply the model in a specific insider misuse case.

The initial section of this Chapter explained that the purpose of the model is to associate user attributes as well as file, network and command execution events to the likelihood of the occurrence of certain incidents. Thus, after entering the user attributes and training the Fsophistication function, a single signature that combines file, network and command execution threat prediction data is produced to address a particular incident. Obviously, a certain number of incidents are going to be addressed and hence there is going to be a certain number of signatures defined. The following algorithm then summarises the execution of the model:

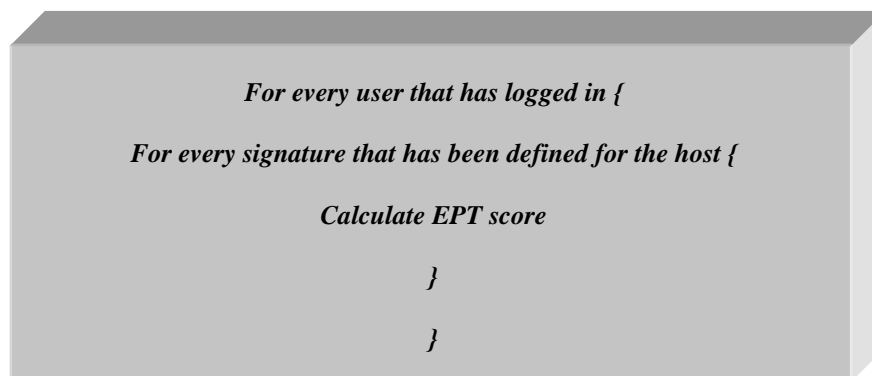
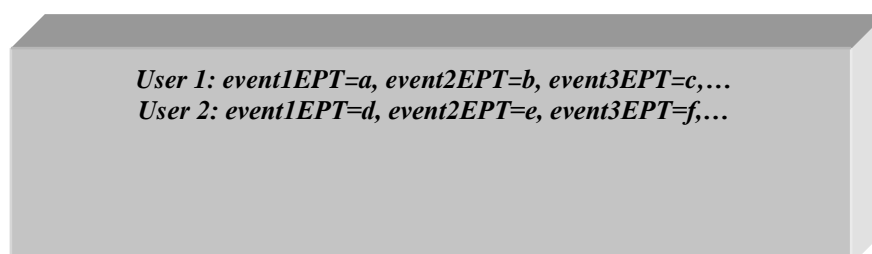


Figure 6.22: ITPM operation scheme

It should be noted that a signature per host scheme is suggested. Although computer systems with the same hardware and software share common properties, each host might have different users or configuration options that will affect the likelihood of certain threats in different ways. The end result of this operation scheme (Figure 6.22) is a series of user EPT scores associated to their respective threat signatures, as shown in Figure 6.23. The system administrator/security specialist can then sort the results per user, EPT value or query the results in any way he wishes, in order to monitor emerging threats on the computer system. Chapter 7 will describe in detail the ways of organising the storage and the query of these results.



.

User n: event1EPT=x, event2EPT=y, event3EPT=z,...

Figure 6.23: The end result of the ITPM model

The aforementioned ITPM operation scheme requires a mechanism for combining the file, network and command execution signatures into a single signature scheme. Figure 6.24 presents the details of this multi-level signature scheme.

```

#Header
ipaddress, targetos,day,month,year
usercategory,reason,keyword1,keyword2,keyword3
WCrole,WCsysadm,WCcriticalfiles,WCutilities,WCphysicalaccess,WFsophistication,WFileops,WFnetops,WFexecops
#Fileops
FileStatement1, FileStatement2, FileStatement3, ..., FileStatementn
#Netops
NetStatement1, NetStatement2, NetStatement3, ..., NetStatementn
#Execops
CcommandcodeAargument1argument2...-##8#

```

Figure 6.24: The multi-level signature scheme

At the header section, the signature stores information employed by the ITPM management system such as the IP address of the host it was created for ('ipaddress'), the target Operating System platform ('targetos'), the date of its creation and a number of keywords related to what type of incident the signature addresses. In particular, 'usercategory' defines what type of users the signature refers to in relation to their level of sophistication as defined by figure 5.2 of the Insider Misuse taxonomy. The 'reason' indicator can be assigned either the value 'accidental' or the string 'intentional', to indicate whether the signature tries to predict non-intentional or intentional incidents. Finally, keywords 1-3 address the nature of the incident (for example "password cracking", "proprietary information theft").

The eight W indicators of the following line prescribe the Weight Matrix (Figure 6.1) for the signature. The manifestation of insider misuse incidents occurs in different ways and hence the importance of threat indicators at the file, network and command execution levels should also be different. Therefore, it is important for the model to have the flexibility of modifying the weightings of the various components per signature, if the signature author wishes to perform such modifications. Section 6.3 presented the default Weight Matrix for all signatures. If the default one is being used for some or all of the weight indicators, then each of these weight indicators contains the string 'default'. In the case of a

mixture of default and modified weight indicators, the only requirement is that the weights must add up to one hundred points.

The rest of the encoding scheme contains the actual signatures for the file, network and command execution level indicators that were explained in the previous sections of the chapter. This concludes the explanation of the multi-level signature designs and the remaining of this section will focus on how this scheme can be applied in the specification of example threat prediction scenarios.

One common problem of legitimate user misuse is the installation of Peer-to-Peer (P2P) file sharing software on corporate computer systems, falling under the unauthorized software installation insider misuse category. Although this incident category was not one of the most common ones in the Insider Misuse survey (chapter 4), it is often linked to the sharing of pirate and pornographic material [97]. The dissemination of pornographic material is frequently encountered in computing infrastructures and hence the act of installing P2P applications represents a realistic scenario for the production of a misuse threat prediction signature. The way of thinking for producing such a signature is presented in the following paragraphs.

In the UNIX/LINUX world of operating systems, mutella [98] is a frequently used P2P application. It is normally downloaded as a tape archive (tar file) that is normally compressed (gz extension) and is available for download from various websites on the Internet. As a result, there are three actions that the legitimate user needs to perform, in order to complete the misuse act.

- i) Locating a source for downloading the application.
- ii) Download the application
- iii) Extract the application from its package format
- iv) Execute the application

Step iv) marks the completion of the misuse act and thus the threat prediction factors should be derived by the first three steps. The production of the signature requires the interpretation of these three steps into file, network and command execution operations. Establishing the command execution signature is a relatively simple task. The reader might recall from previous sections that there is a host command-

logging facility on each computer host (more details about the facility are given on chapter 7). During the execution of the third step, the command logging facility produces a log displayed in figure 6.25 below:

```
Jun 13 12:06:02 kerberosdev snoopy[23252]: [georgios, uid:502 sid:23213]: /bin/tar xvfz mutella-0.4.3.tar.gz
Jun 13 12:06:08 kerberosdev snoopy[23255]: [(null), uid:0 sid:23162]: /bin/grep georgios
Jun 13 12:06:58 kerberosdev snoopy[23256]: [georgios, uid:502 sid:23213]: /bin/ls -F --color=auto
Jun 13 12:07:07 kerberosdev snoopy[23257]: [georgios, uid:502 sid:23213]: /bin/ls -F --color=auto
Jun 13 12:07:14 kerberosdev snoopy[23258]: [georgios, uid:502 sid:23213]: /bin/vi README
Jun 13 12:07:20 kerberosdev snoopy[23259]: [georgios, uid:502 sid:23213]: /bin/ls -F --color=auto
Jun 13 12:07:28 kerberosdev snoopy[23261]: [(null), uid:0 sid:23162]: /bin/grep georgios
Jun 13 12:08:14 kerberosdev snoopy[23262]: [georgios, uid:502 sid:23213]: ./configure
.
.
Jun 13 12:14:05 kerberosdev snoopy[24545]: [georgios, uid:502 sid:23213]: /usr/bin/make
.
.
Jun 13 12:14:09 kerberosdev snoopy[24546]: [georgios, uid:502 sid:23213]: /usr/bin/make install
.
.
```

Figure 6.25: Command line logging data as a result of the mutella tape archive extraction

The entries in bolded characters represent the commands that are unique to the operation and hence they should constitute the command-line execution level component of the signature. The configuration and compilation of the application produces a fair amount of output that has been excluded. The exclusion was on the grounds of commands that might be user specific and hence impede the generic character of the signature. Consequently, if someone extracts the date, user and other header data and preserves the sequence of the entries in bold (as discussed in section 6.3.2.3), the following commands constitute the execops data level of the signature:

```
/bin/tar xvfz mutella-0.4.3.tar.gz
/bin/vi README
./configure
/usr/bin/make
/usr/bin/make install
```

However, one might argue that the user might employ a different sequence of commands to uncompress and extract the mutella archive. For example, instead of using the tar command with the xvfz switches, in order to uncompress and extract the tar archive in one step, he might have followed different steps:

```
/bin/gunzip mutella-0.4.3.tar.gz
/bin/tar xvf mutella-0.4.3.tar
/bin/vi README
```

```
./configure  
/usr/bin/make  
/usr/bin/make install
```

Additional combinations do exist and hence the previously presented command line sequences should both be part of the signature. Section 6.3.2.3 explained that it is possible to provide many command sequence variations by using the OR operator and hence the execops level component of the signature would have been encoded in the following form.

```
#Execops  
C/bin/tarAxvfzmutella-0.4.3.tar.gz##8#C/bin/viAREADME##8#C./configureA##8#C/usr/bin/  
makeA##8#C/usr/bin/makeAinstall##8# OR  
C/bin/gunzipAmutella-0.4.3.tar.gz##8#C/bin/tarAxvfmutella0.4.3.tar##8#C/bin/viAREADME  
##8#C./configureA##8#C/usr/bin/makeA##8#C/usr/bin/makeAinstall##8#
```

Having tackled the command line signature component, the installation of the mutella application will leave certain traces at the file and network levels. Using the Internet to locate a software repository, in order to download the application normally involves the acts of quering a search engine and then connecting to one or more web or FTP servers to retrieve the tape archive for the application. On a standard system, the act of downloading the application can actually take seconds or a few minutes at most, depending on the bandwidth capacity of the established connection.

An average system creates and destroys many endpoints. Although logging facilities for network endpoint creation and destruction are feasible they are not often employed by standard operating system utilities. Thus, searching for a network endpoint that exists for a few seconds or minutes and then leaves no standard traces does not represent a reliable threat prediction data source in this particular case.

Instead, one can focus on file operations. In fact, file-level evidence that could be used to audit certain network events is possible in this case. Many Internet Web browsers optionally employ a 'history' file, where they keep track of all the Uniform Resource Locators (URLs) that the user visits. The implication of this feature in this case is that one can intercept evidence of Search Engine activity related to searches for the 'mutella' application, as well as visits to web pages where one would expect to find a mutella tape archive available for download.

For instance, the Netscape World Wide Web browser [99] maintains the 'history.dat' file for every computer system user. This file holds all the URLs that a particular user has visited for a period of time. Microsoft's Internet Explorer and other World Wide Web browsers exhibit similar functionality. Hence, if a user searches by using the keyword 'mutella' on a number of different search engines (Google and Yahoo for example), the file will contain entries as the ones shown below:

http://www.google.co.uk/search?q=mutella&ie=UTF-8&hl=en&meta=

http://uk.search.yahoo.com/search/ukie?p=mutella&y=i&ei=ISO-8859-1&fr=fp-tab-web-t&cop=mss&tab=

Both URL encoded entries contain the strings 'search' and '=mutella'. These would point to a number of links that would eventually link to a download link such as the one below:

http://prdownloads.sourceforge.net/mutella/mutella-0.4.3.tar.gz?download

Hence, one can now establish the file-level component of the multi-level signature, assuming that the user browses the Internet with Netscape Navigator:

existsf:<history.dat>:<dat><s>:<s>:<*search*=mutella* OR *mutella*>:<\$HOME> OR

existsf:<mutella*.tar.gz>:<tar or tar.gz>:<s>:<s>:<\$HOME, /usr/src> OR

existsd:<mutella*>:<s>:<s>:<s>:<u>{:existsf:<mutella>:<binary>AND

existsf:<AUTHORS>:<asciitext>:<contains:Mutella Project>}

This particular multi-level signature will not employ network-level components. Hence, the EPT component Weight Matrix needs to be re-defined, in order to accommodate for the absence of network-level data and re-distribute their weight to file and command-line execution data. This will be indicated amongst other data in the header of the signature (Figure 6.26).

#Header

192.168.2.33, Linux testbox 2.4.21-15.ELsmp #1,23,05,2003

sysmasters,intentional,unauthorised software installation,mutella,P2P

6,6,6,6,10,30,0,30

#Fileops

existsf:<history.dat>:<dat><s>:<s>:<*search*=mutella* OR *mutella*>:<\$HOME> OR

existsf:<mutella*.tar.gz>:<tar or tar.gz>:<s>:<s>:<\$HOME, /usr/src> OR

```

existsd:<mutella*>:<s>:<s>:<s>:<u>{:existsf:<mutella>:<binary>AND
existsf:<AUTHORS>:<asciitext>:<contains:Mutella Project>}
#Netops
#Execops
C/bin/tarAxvfzmutella-0.4.3.tar.gz##8#C/bin/viAREADME##8#C./configureA##8#C/usr/bin/
makeA##8#C/usr/bin/makeAinstall##8# OR
C/bin/gunzipAmutella-0.4.3.tar.gz##8#C/bin/tarAxvfmutella0.4.3.tar##8#C/bin/viAREADME
##8#C./configureA##8#C/usr/bin/makeA##8#C/usr/bin/makeAinstall##8#

```

Figure 6.26: The completed multi-level signature for predicting the P2P mutella installation

This completes a practical example that aims to illustrate how one can translate the first steps of an insider misuse act to system-level requirements, in order to produce a threat prediction multi-level signature.

6.5 Conclusions

This chapter presented the Insider Threat Prediction Model, the most important component of this research project. The model relates the role and access privileges of users (referred to as user attributes), as well as their on-line behavioral characteristics (legitimate user sophistication and file, network and command line execution data) to the potential (not the true probability) of occurrence of certain events. Although experimental results were discussed, this chapter focused on the theoretical principles upon which the design of the model is based. The next chapter will address the development of a prototype system, in order to provide additional details about the implementation and further refinement of the model's operation.

CHAPTER 7

THE ITPM SYSTEM ARCHITECTURE

The thesis has presented so far all the important components of the Insider Threat Prediction Model. However, the prototype implementation of this model on a real world Operating System will require a supporting architecture that will address a certain number of issues such as:

- A) The efficient storage and organization of the model's data (signatures, user session data, operational parameters).
- B) The addressing of the performance problems the model might encounter in large computational environments.
- C) Cross-platform compatibility issues, as the architecture will inevitably need to be implemented in more than one Operating System, beyond the prototype level.
- D) Speed of development, as the time scales for this research project were limited to less than 12 months.
- E) The data security of the data and the associated operations that modify them is enhanced as much as possible in the prototype system.

This chapter will start by explaining the choice of the development environment tools. Throughout the body of the thesis, there are references to LINUX/UNIX system and the PERL programming language. These choices were never justified properly and thus section 7.1 will explain the reasons for making these choices. Section 7.2 will address the issue of data organization in the ITPM model. A discussion of a suitable client/server architecture to implement the model is the subject of section 7.3. Finally, the chapter concludes with the task of addressing performance and scalability issues related to the implementation model on a real-world operating system.

7.1 The ITPM prototype development environment

In the process of implementing an ITPM prototype engine on a real-world operating system, one has to choose a particular operating system and programming language to implement the code of the system. These choices are of fundamental importance, since they define what can be done and also how quickly certain goals can be performed.

Operating System	Cost	Source Code availability	Hardware compatibility	Supercomputer infrastructures
LINUX	Free	Excellent	Very Good	Supported
FreeBSD	Free	Excellent	Not so good	Supported
Windows 2000/XP	High	Non existent	Excellent	Non existing

Table 7.1: Operating System (O/S) selection factors

Table 7.1 displays a selection of widely employed O/S choices in the Intel/AMD 32-bit microprocessor arena. Starting with the Microsoft Windows family of products, they exhibit three fundamental problems. The first concerns the unavailability of the O/S source code with secondary problems concerning the price of the base O/S and the development tools and lastly the lack of the Operating System’s suitability for a supercomputing environment during the development phase of the research project [100]. The importance of supercomputing facility support is discussed in section 7.3.

Amongst LINUX [101] and Free/BSD [102], the choice was based on the fact that the latter O/S does not provide as good support for commodity PC hardware as LINUX does. Although both LINUX and Free/BSD target the personal computers and offer (at no additional cost) a wide range of programming language/compiler by default, LINUX has a larger community of developers when it comes to device drivers and hence it has become substantially more user friendly than Free/BSD. In fact, LINUX has today grown into a commercially acceptable O/S addressing both the server [103] and desktop computing market [104]. For all these reasons, LINUX is the best O/S choice for the ITPM prototype system.

The Practical Export and Reporting Language PERL, the Tool Command Language TCL and the Python Programming Language were considered as the candidate programming languages. All of these choices constitute scripted programming languages [105] often employed in the prototyping of systems and all of them are cross-platform oriented, in order to reduce the time to develop basic prototype tools.

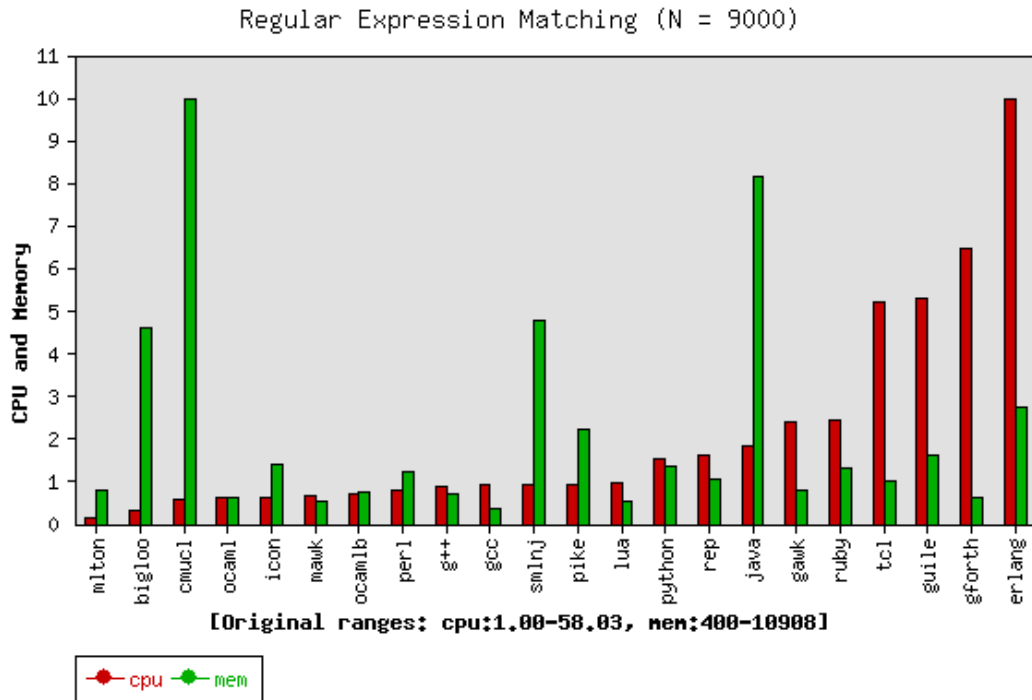


Figure 7.1: Regular expression performance amongst various programming languages

The graph of figure 7.1 indicates the results of a test that measures the computational efficiency of the pattern matching (regular expression) engines of various programming languages [106]. The test included string extraction operations according to certain regular expression criteria from an input file. The x-axis represents various programming languages, whereas the y-axis represents a normalized computational cost in terms of CPU execution time and memory consumption footprint. Higher costs indicate poorly performing languages.

Based on these assumptions, the graph of Figure 7.1 indicates that the PERL programming language offers the best regular expression features in terms of computational performance amongst the three originally selected languages (PERL, Python, TCL). For these reasons, PERL was the best choice for this research project.

Figure 7.1 also indicates the performance supremacy of compiled programming languages such as C and C++. It is commonly accepted that programming languages that produce native machine code have always better performance than interpreted languages. However, the prolonged development time they introduce make software engineers choose to prototype their systems in scripted languages and then build the performance intensive parts in compiled languages [105].

7.2 Organizing the ITPM data

Insider Threat Prediction Model Database Schema

Users	
PK	<u>userid</u>
	unixuid unixgid adsid firstname middlename lastname homeonhost Crole Csysadm Ccriticalfiles Cphysicalaccess Fattributes allfortargetos usercategoryflag

signatures	
PK	<u>signid</u>
	targetos hostip Crole creationday creationmonth creationyear reason keyword1 keyword2 keyword3 signfile

Events	
PK	<u>eventid</u>
	userid signid Fbreadth Fappscore SCPU SRAM SSIMAPPS Fresutil Fdepth Fsophistication Ffileops Fnetops Fexecops Fbehavior EPT

hosts	
PK	<u>hostid</u>
	hostip targetos signaturedir noofusers usrmd5sum

Figure 7.2: The ITPM database schema

Chapter 6 presented a plethora of data sources including misuse prediction signatures formats as well as user command line data collection structures. All of these data require efficient organization. In particular, it is important that the process of comparing user and system activity against a number of particular misuse signatures should be easy and transparent. Thus, a Relational Database Management

System (RDBMS) should be employed, in order to aid the process of efficient data management. The RDBMS schema of such a database is shown in Figure 7.2.

The schema consists of four tables. The 'Hosts' table keeps track of all the individual computer systems managed by the ITPM system by recording the computer system's IP address (hostip), operating system (targetos) and the system directory where the signatures (for this particular host) are stored (signaturedir). The 'noofusers' column indicates how many users exist in the host, whereas 'usrmd5sum' is used to indicate changes in the user entry database of each computer system.

The purpose of the 'Signatures' table is to allow the ITPM system operator to store and query the produced misuse signatures in an efficient manner. It should be emphasized that the 'Signatures' table does not store the contents of the signature, but only information related to its header (Figure 6.24). As a result, signatures can be queried by one or more qualifiers such as the operating system of the host (targetos), the IP of the host where the misuse signature is applicable (hostip), the user category that they refer to ('Crole' in relation to Figure 5.2), as well as the day, month and year of their creation. The 'reason' column indicates whether the signature refers to accidental or intentional misuse. Lastly, the three keywords identify further the scope of the contents of the signature and 'signfile' is the full path to the file that contains the signature in the host signature repository.

The 'Users' table provides a repository of certain legitimate user attributes associated to the ITPM model. Most of the column names are derived by the names of the various ITPM equation parameters, as described in the sixth chapter of the thesis. The rest of the column names are related to operating system user identification data. UNIX-like operating systems always contain a numeric user identification number for the user and the file group that the user belongs to (unixuid, unixgid). In order to preserve cross platform compliance, the database also accommodates Microsoft's Active Directory Security Identifier (SID) as an alternative O/S user identification method. An Active Directory SID is a "unique value of variable length used to identify a user account, group account, or logon session", according to Microsoft's TechNet documentation [107].

The last two column identifiers of the 'Users' table define which signatures are applicable to a particular user. If the 'allfortargetos' flag is set (binary data type), then the user actions will be checked against all the signatures that are defined on the host. Alternatively, if the 'allfortargetos' flag is not set, then the user actions are going to be checked against the host signatures that concern the 'usercategoryflag' he belongs to. This optional feature can help the ITPM operator reduce the computational resources required for the operation of the model, by refining the number of signatures that need to be checked.

Finally, the 'Events' table contains a collection of user EPT score values that originate from the checking of user accounts against the defined insider misuse signatures. Figure 6.22 explained the basic operation of the model by indicating that every active user is checked against a number of signatures and hence this table is the most frequently modified one during the operation of the system. One can observe that where the 'Users' table contains mostly the parameters of the users that are modified less often ($F_{\text{attributes}}$), the 'Events' table holds the user attributes that are more dynamic in nature (F_{behavior}).

The derived database schema is not normalized and its referential integrity is solely dependent on the ITPM system application code. For instance, if a particular host is removed from the 'hosts' table, the ITPM system is responsible for removing the associated entries from the 'Signatures', 'Users' and 'Entries' table. The same effect could be achieved by employing foreign keys, a feature often employed in Relational Databases. However, the choice of not employing foreign keys and other RDBMS driven referential integrity mechanisms was an intentional one, in order to make the task of implementing the scheme to various relational databases easy. The implementation of these schemes amongst the various available databases may differ substantially and it would reduce the portability of the ITPM system.

The MySQL RDBMS system [84] was chosen to implement the database schema. Section 6 of Appendix E contains sample code for the creation of the table scheme. MySQL is an Open Source, cross platform RDBMS that is widely known for its excellent performance. PostgreSQL [108] is also a popular Open Source RDBMS. Although PostgreSQL supports a larger array of relational database features than MySQL, for simple SQL queries the latter RDBMS is faster and more computationally

efficient than PostgreSQL [109]. In addition, the compatibility between MySQL and PERL has been proven throughout the development world.

Commercial products such as Oracle [110], Sybase [111] and IBM DB2 [112] were excluded due to licensing costs. Although these products are feature rich and scalable, the extra features they offer on performance and scalability are not really needed in the early ITPM prototype platform.

7.3 The ITPM Client-Server architecture

Having addressed the issues of selecting a suitable O/S and a programming language to develop the code, as well as a way of organizing the ITPM data, the next logical step in the process of building a prototype system is to consider how the functional blocks of the system fit together, in order to address the basic requirements of scalability, operational integrity and cross-platform compatibility. Figure 7.3 provides the functional diagram of the ITPM system.

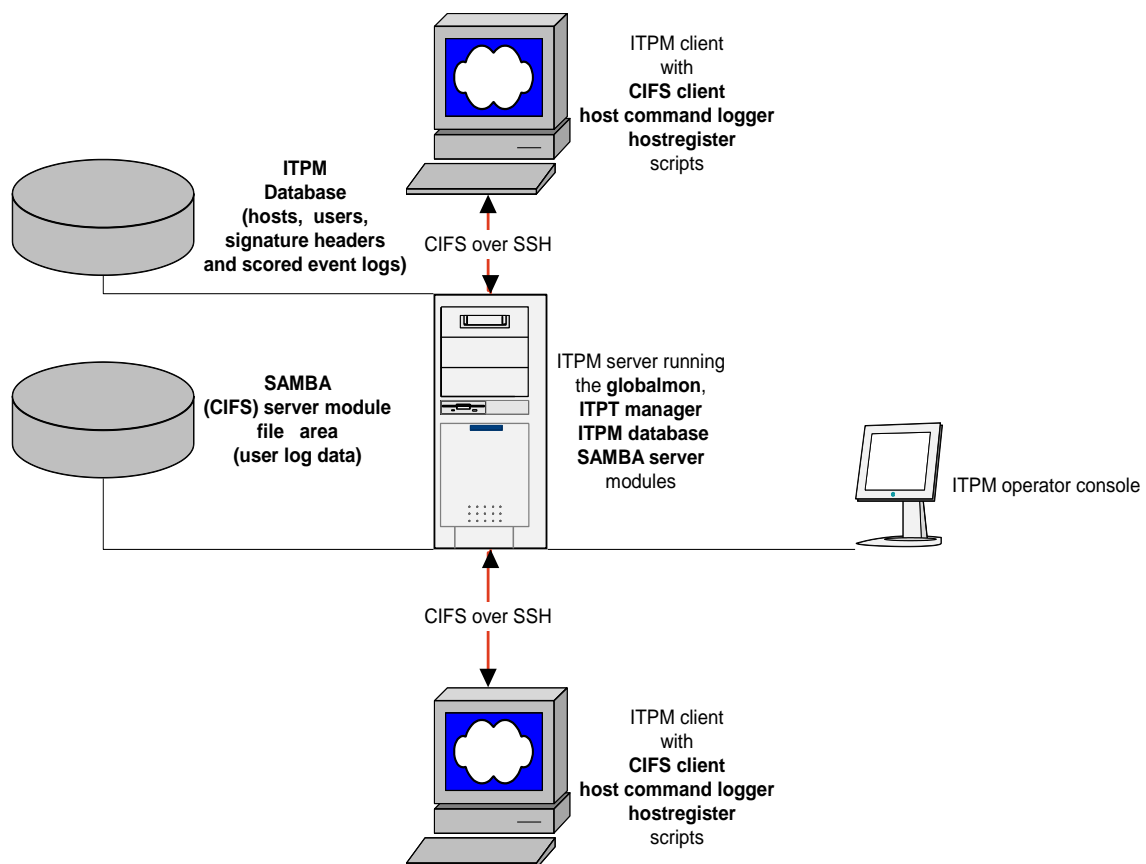


Figure 7.3: The ITPM Client/Server architecture

The basic reasoning for designing this architecture is twofold. One of its aspects emerges from the need to place the most computing intensive tasks on a dedicated machine. Chapter 6 elaborated on the performance characteristics of the various ITPM algorithms showing that some of them can create a serious contention issue with regards to computational resources. Thus, with the cost of 64-bit computing hardware falling [113], instead of placing the burden on the resources being monitored, it is better to purchase a dedicated server system which will take care of the most computationally intensive aspects of the system.

Another aspect of the client/server scheme concerns the operational integrity of the model. The task of leaving important operational data (such as the ITPM database, the stored signatures) spread around various systems creates additional data security complications, depending on who controls the various machines and his intentions. A central repository of these data is a better idea in terms of data security and is also a factor that eases the maintenance of the system. For instance, in order to ensure data availability, it would be much easier to backup databases and user attributes from a central host rather than a large number of them.

The data security requirement aspects are also reflected on the communication channels between the ITPM server and its clients. All data exchanges for program execution and file-system access are performed via Secure Shell (SSH) protocol sessions [78]. A more application-orientated description of the SSH protocol is given in [114]. The SSH protocol provides a flexible and secure mechanism for connecting data streams together amongst machines connected via the TCP/IP protocol suite (the requirements listed in section 6.3.2.2 explained why TCP/IP is chosen as the preferred protocol for the ITPM architecture). The adequacy of the protocol's security functionality is justified by certain features the protocol has to offer such as:

- SSH encrypts the data stream amongst two machines and thus prevents unauthorized parties from eavesdropping at either end or anywhere in the middle of the communication path. A variety of encryption algorithms can be employed such as triple DES [115], certain versions of the Advanced Encryption Standard [116], the Blowfish cipher [117] and others.
- SSH offers authentication by means of the widely employed Diffie-Hellman public-key distribution method [118]. As a result, a communicating host can verify the true identity of its

peers. This functionality is highly desirable in the ITPM architecture as it prevents a variety of identity spoofing techniques, as well as the ‘man-in-the-middle’ attack [119]. Although the original version of the Diffie-Hellman public key exchange [118] was vulnerable to ‘man-in-the-middle’ attacks, the SSH protocol uses the revised authenticated version [119].

- SSH provides data stream integrity. Every single message of the data stream that gets received by the other side is checked for completeness and malicious alterations. The message integrity feature prevents replay attacks [114], where a session is replayed to cause the same action to be repeated for the purposes of bypassing the security defenses of a system.

The flexibility of the SSH protocol is specified by its ability to encapsulate other application-level protocols within its data stream. This feature is called ‘tunneling’ and allows insecure protocols to be employed securely in an IT infrastructure. This is the case with the Server Message Block protocol [120], whose core was developed to share printers, files and serial ports amongst computers. In 1996, Microsoft has extended SMB and renamed it to ‘Common Internet File System’ (CIFS). However, despite the CIFS additional features, the protocol suffers from many security deficiencies [121]. Although some of the vulnerabilities mentioned in [121] have been addressed, the protocol is still considered insecure.

Despite the security issues, the CIFS protocol eventually became the ubiquitous interface to provide common file system and printing services amongst UNIX and Microsoft Windows-based computers. It is, hence, an excellent choice with regards to cross-platform compatibility. Therefore, if one combines the CIFS and SSH by means of encapsulating CIFS traffic via an SSH tunnel [114], he will achieve a cross-platform solution that also addresses the security concerns of the core CIFS protocol.

The ITPM server host (Figure 7.3) runs a ‘Samba’ CIFS server [122] on the LINUX O/S. The purpose of this server is to host directory areas that can be mounted by the ITPM clients. The clients can then deposit the various user data (command execution logs, misuse signatures) on these directories. Server processes can also access these directory areas to analyze the collected data and populate the ITPM Relational Database (MySQL) (as described in section 7.2) that runs also on the server host. Lastly,

the ITPT manager is a Graphical User Interface application that coordinates all the system utilities and allows the ITPM operator to interface with the system.

The ITPM client runs a series of monitoring scripts under the control of the server-based ITPM manager application. The 'globalmon' script is responsible for monitoring all the active users of the system. An 'active' user is one that owns one or more processes on a client system. Hence, the script maintains an up-to-date list of all the active users on a system and then executes a series of data logging functions for every user of that list. The 'globalmon' script is also responsible for secondary house keeping functions, such as the task of making sure that the client host is properly registered with the ITPM server.

The 'host command logger' is a very important system-level application that provides a log of all the commands executed by every user of a client host. Section 6.4 of the previous chapter assumed the existence of this facility and figure 6.25 provided a sample log generated by this facility. On a UNIX/LINUX operating system, this functionality can be achieved by intercepting all the execve system calls [123].

System calls are consistently defined functions that software applications can invoke whenever they wish to perform an Operating System function. These functions are often combined together into an Applications Programming Interface (API) library. File access, memory allocation and release, as well as the starting and stopping of other applications are some characteristic operations that are performed by means of invoking certain API system calls from a software application. In the host command logger facility's case, the goal is to keep a record of all the commands executed by all users. An execve system call wrapper is a small program that intercepts every single execve system call executed by the O/S and places the output into a human readable file. The project employed 'snoopy' [124], an open source execve wrapping utility. The source code of this utility is given in Appendix E (section E5).

The 'hostregister' client script is responsible for registering a client with the ITPM server (Appendix E section 7). The registration process involves the enumeration of the operating system and user area executable commands as described in section 6.3.2.3, the generation of the necessary authentication

credentials for the purposes of communicating via the SSH protocol and procedures to initialize the ITPM database tables with user and host related data. Finally, the ‘createsignature’ script (Appendix E section 4) is also a client-based utility that is responsible for constructing the multi-level insider misuse prediction signature (section 6.4). It is invoked by the ITPM manager application, however it operates on the ITPM client OS and creates host-specific signatures.

The source code of all the aforementioned scripts of both the client and server hosts is provided in Appendix D of the thesis.

7.4 ITPM system scalability considerations

The previous sections of the Chapter presented the design choices for constructing an ITPM prototype system. Figure 7.3 presented a client/server scheme where all the threat prediction computations are performed on a single ITPM server. Depending on the computational power of the ITPM server and as the number of hosts/users increases, there are two types of limits that are likely to create a bottleneck. Both of them can seriously impede the operation of the proposed architecture.

The first type of bottleneck concerns the number of monitoring processes that are started in the ITPM server host. Each client host executes an instance of the ‘globalmon’ script. Each instance of the script might invoke in series further applications, in order to monitor the list of active users. In addition, a single instance of the ‘globalmon’ script and all of its child processes can demand several tenths of megabytes of RAM. As the amount of CPU time and RAM memory on the server system is finite, there is clearly a limit where the system will refuse to execute further instances of the script or allocate more memory for threat prediction computations. Long before that limit is reached, a deterioration in the responsiveness of the system is expected that might render the server unusable.

Bottlenecks are also expected to occur in the operation of the ITPM database. The execution of the various scripts results in a number of database queries (reads and writes). The number of queries increases in proportion to the number of registered hosts. Heavily utilized RDBMS engines draw quite a lot of CPU and RAM computing resources, representing a second important point of resource contention in the ITPM server host.

In order to prove these predictions, a load testing plan was devised, in order to simulate the load of several instances of the globalmon script on the ITPM server host. There were no provisions to test the system on a large infrastructure, so the computational impact had to be estimated by means of load testing simulation. Each instance of the globalmon script operated on 100 virtual users (all running on the same client host) and with just 20 sample signatures in the ITPM database. The results of this testing scheme on a dual Pentium III 1GHz system with 2 Gb of RAM (ITPM server) are shown in the graphs of figures 7.4 to 7.6.

The graphs indicate clearly that this particular ITPM server was no capable of withstanding more than 40 instances of the script under simulated conditions. This means that the single server could roughly monitor 30-40 multi user hosts. The invocation of more instances of the script caused serious swapping activity on the system and eventually the machine ran out of memory with noticeable performance degradation, well before the launch of the 40th globalmon script instance. This performance would be acceptable in small business and research environments, but is clearly unsuitable for medium to large enterprise computing environments.

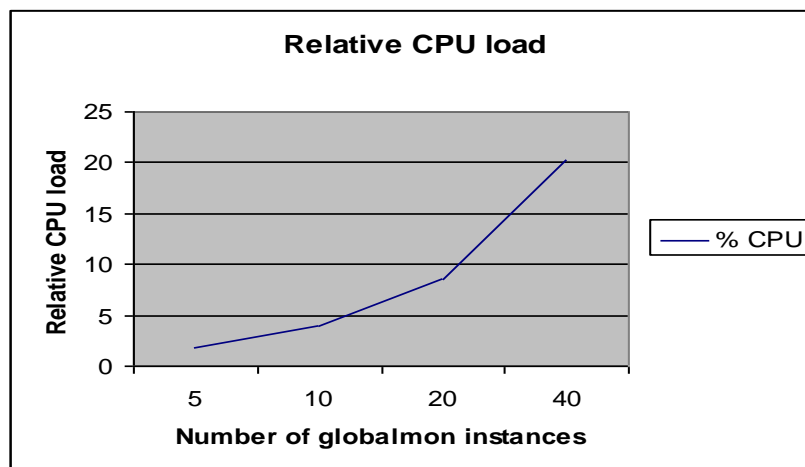


Figure 7.4: Single ITPM server CPU load

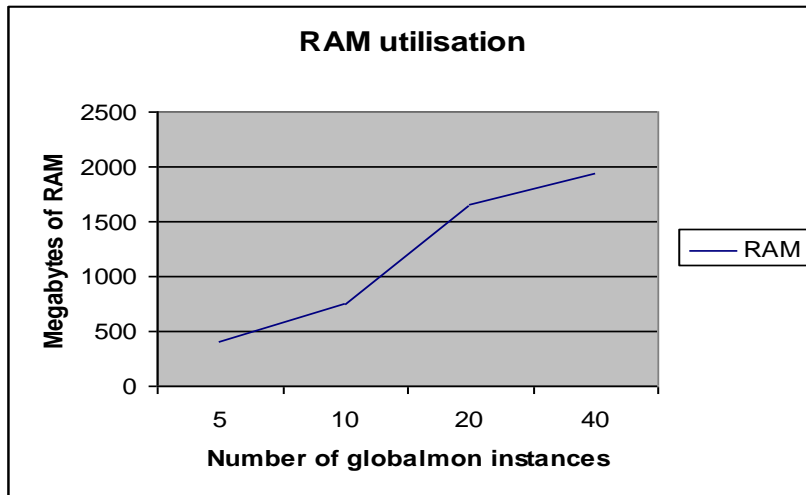


Figure 7.5: Single ITPM server RAM utilisation

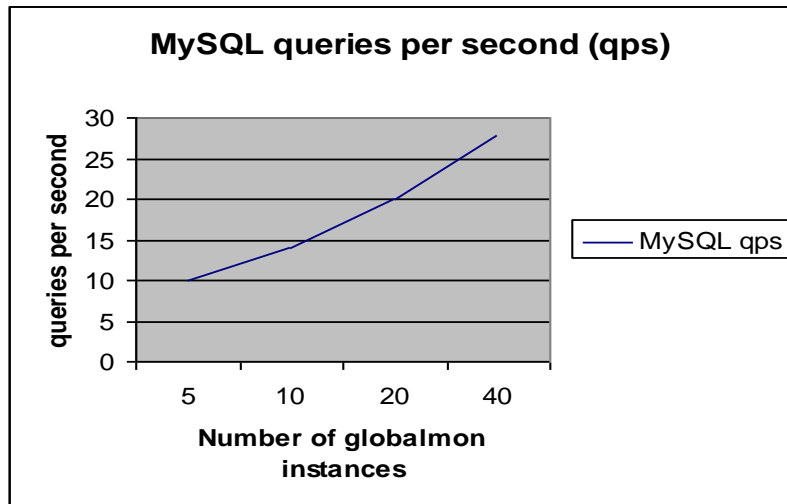


Figure 7.6: Single ITPM server MySQL load

If one wishes to address the previous resource contention issues, he will have to expand the computational resources by adding more RAM and a greater number of CPUs to the ITPM system. This goal can be achieved by employing Multi-Processor systems [125]. A Multi-Processor computer system consists of a hardware architecture that interconnects two or more CPU, as well as a number of RAM memory chips, in order to process different instructions simultaneously and increase the overall computational throughput. The way of interconnecting the CPU and memory modules distinguishes Multi-Processor systems into ‘tightly coupled’ and ‘loosely coupled’ systems. This distinction is very important because it affects its cost, performance and the system maintenance complexity.

A ‘tightly coupled’ Multi-Processor design is often constructed as a Symmetric Multi Processing (SMP) system. The term ‘symmetric’ implies that the interconnected CPUs are identical in terms of hardware architecture. It also means that the O/S should not have any bias towards allocating specific programs to particular CPUs. All CPU modules are considered equally capable of executing any type of application. A basic diagram of an SMP dual CPU system is illustrated in figure 7.7.

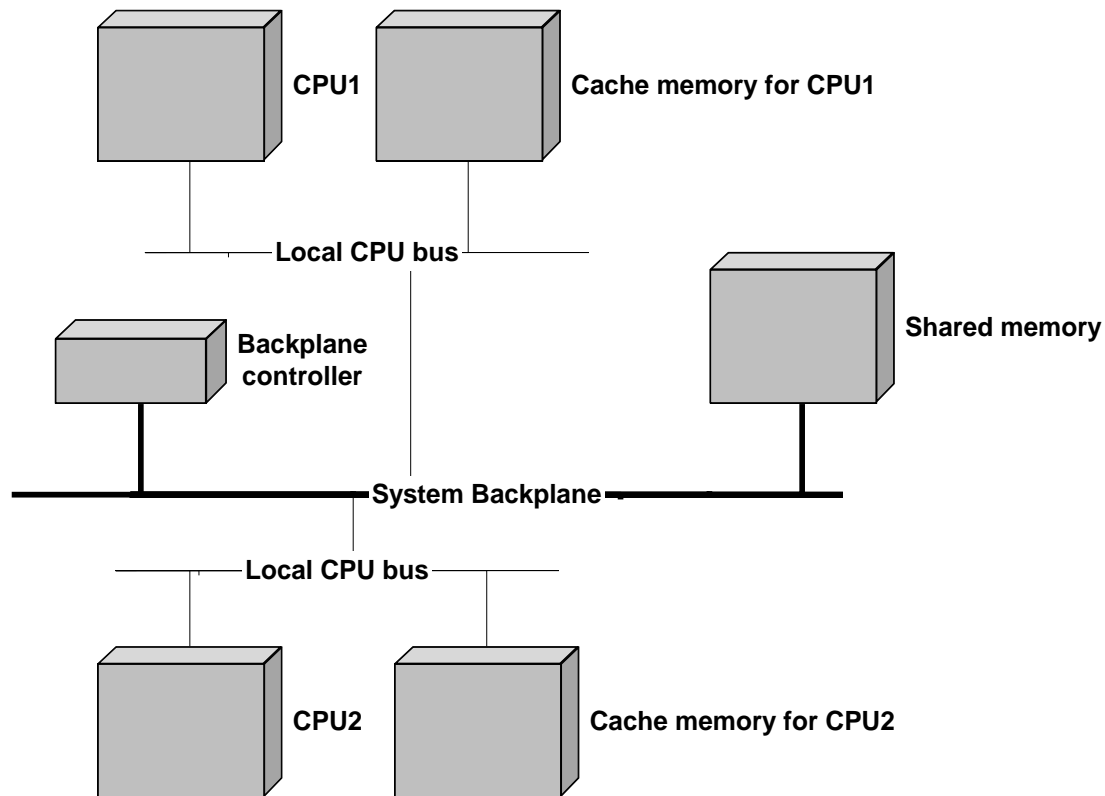


Figure 7.7: SMP computer system Functional blocks

The ‘system backplane’ is a bespoke interconnection bus that is highly optimized for inter-CPU communication and memory access. The interconnected CPUs can access a global ‘shared memory’ area and they also maintain their own cache memory. The operation of the ‘system backplane’ bus is regulated by the ‘backplane controller’ module.

On the other hand, ‘loosely coupled’ MP systems can be constructed by replacing the highly optimized ‘system backplane’ with a more conventional Data Network technology. In this case, the CPU and memory modules can be ordinary uni-processor computer systems, and the interconnection medium is normally provided by a Data Network technology such as Gigabit Ethernet, the more expensive Myrinet and other experimental interconnects such as Infiniband [126]. A loosely coupled MP system

does not have explicit hardware support for shared memory. The memory resources are distributed and specialized software is required to emulate a shared memory resource area.

Each of the aforementioned MP architectures has its own strengths and weaknesses, depending on the application that one chooses to run. SMP systems exhibit excellent reliability and the minimum amount of system administration maintenance. They also provide excellent performance for applications that require a lot of interaction between them. This is because of the highly optimized system backplane which reduces the latency amongst communicating CPUs and their explicit shared memory hardware support.

However, these features make SMP systems very expensive. Typical prices require several hundreds of thousands of pounds for the purchase of CPU systems that have more than 4 CPUs and this can be prohibitive for small budgets. Scalability (the maximum amount of processors that can be fitted on the backplane) is also limited beyond 256 processors at the time of writing.

In contrast, loosely coupled systems are most efficient when the applications require minimum interaction amongst them. Conventional Data Network technologies such as Gigabit Ethernet are still not able to offer the low inter-CPU communication latencies of SMP system backplanes. A 'Myrinet' interconnection improves further the communication latencies of loosely coupled systems in relation to Gigabit Ethernet [126] at a higher cost, but even that technology is still not efficient as an optimized SMP backplane.

Because loosely coupled systems are often build by commodity hardware components based on hardware manufacturers such as Intel and AMD, they are orders of magnitude cheaper than SMP systems with the same amount of CPU and memory modules. Moreover, because commodity hardware is standardized, it is possible to construct a loosely coupled system from heterogeneous computer systems. This property can maximize the exploitation of computing resources inside an organization and had created the field of commodity HPC clusters. Commodity clusters are able to scale into thousands of processors. They require a larger amount of system administration maintenance but they constitute the cheapest way to build small or large scale HPC infrastructures.

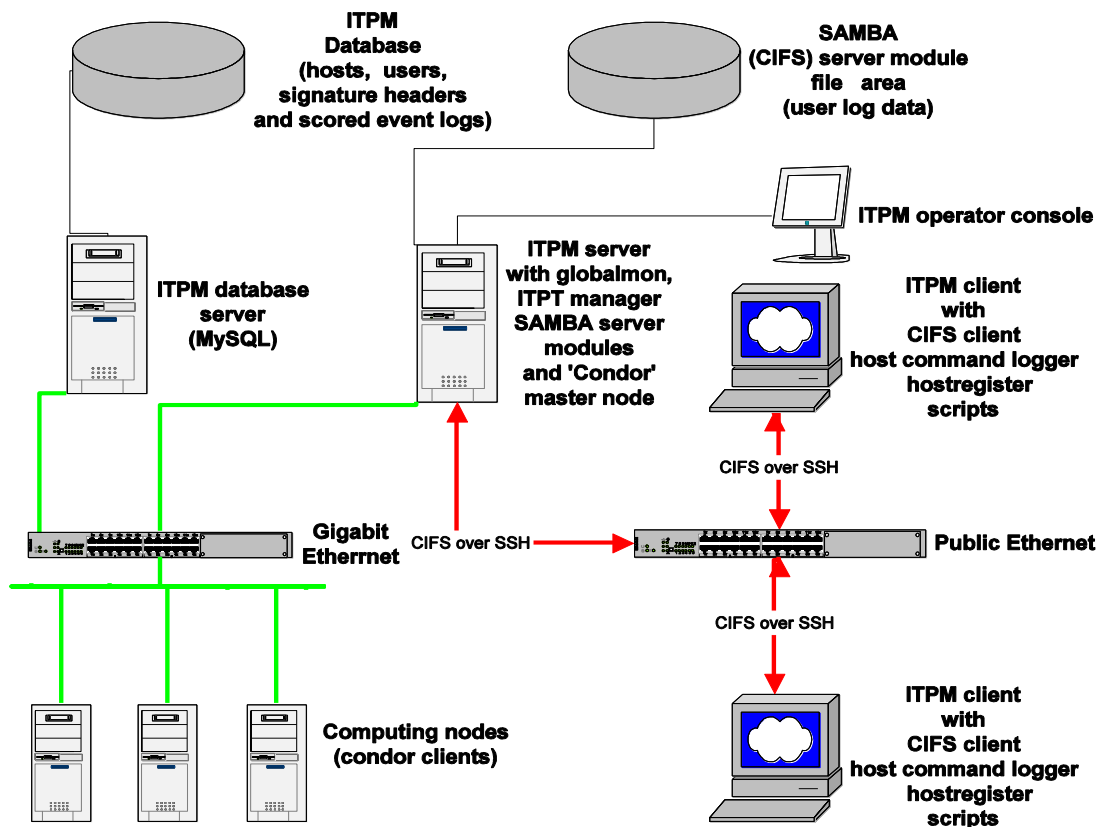


Figure 7.8: Scalable ITPM client/server architecture

Figure 7.8 illustrates a loosely coupled system based on a commodity cluster distributed processing paradigm. This design refines the original client/server schema and allows the ITPM architecture to scale for monitoring large computing infrastructures. The computing nodes (bottom left) are used as additional CPU and RAM memory resources to execute instances of the globalmon script in parallel. The computing nodes and the ITPM servers are connected by means of a dedicated restricted Gigabit Ethernet network segment (green connector lines).

The ITPM database has been placed on a separate server, due to its intensive computational requirements. Hence, one ITPM server is dedicated to the IPT manager application and the allocation of instances of globalmon scripts to the computing nodes, whereas the ITPM database server can offer a dedicated SMP architecture to scale the capacity of the RDBMS application. Database manufacturers such as Oracle [110] and (recently) MySQL [84] can offer RDBMS products that are optimized for distributed (loosely coupled) MP systems. However, at the time of writing, these products are relatively untested and hence SMP systems offer a conventional and well-tested option for RDBMS scalability.

After explaining the logic behind partitioning the ITPM server and the making of a commodity cluster that consists of loosely coupled CPU and RAM modules, we also need to select the software that migrates the globalmon scripts from the ITPM server to the computing nodes. Condor [127] is a freely available complex front-end management system for distributing computing jobs to computational nodes. It runs on many UNIX platforms, including the LINUX O/S.

The architecture of Condor is based on a client/server model. The master server (or 'Central manager') is located on the ITPM server. The master server module collects the job requests and then allocates the jobs to computing nodes according to certain computational resource availability criteria. This differs from the pure SMP model, where there is normally no preference on which CPU will be allocated a particular task. Condor could make smart decisions by allocating the next task to the least CPU loaded node or for instance to a node that has a certain amount of free RAM or disk space available. This functionality allows one to have complete control over the way distributed computing resources are utilized.

In order to put these theories into the test, the load testing simulation was repeated with the Condor installed on the ITPM server and two identical nodes (Pentium III 1GHz, 2Gb of RAM) interconnected as shown in Figure 7.8. We then used the condor batch system to migrate instances of globalmon scripts to the three SMP clients. The end result was that we were able to scale the number of globalmon script instances to approximately 120. In addition, Figure 7.9 shows the separate MySQL performance during the execution of the globalmon instances on the commodity cluster. The RDBMS server was now able to execute approximately five times more queries than in the single server scenario. This indicates that the bottleneck in the single server scenario came really from the execution of the globalmon scripts on a single server and that this architecture will scale the ITPM client server model to larger infrastructures.

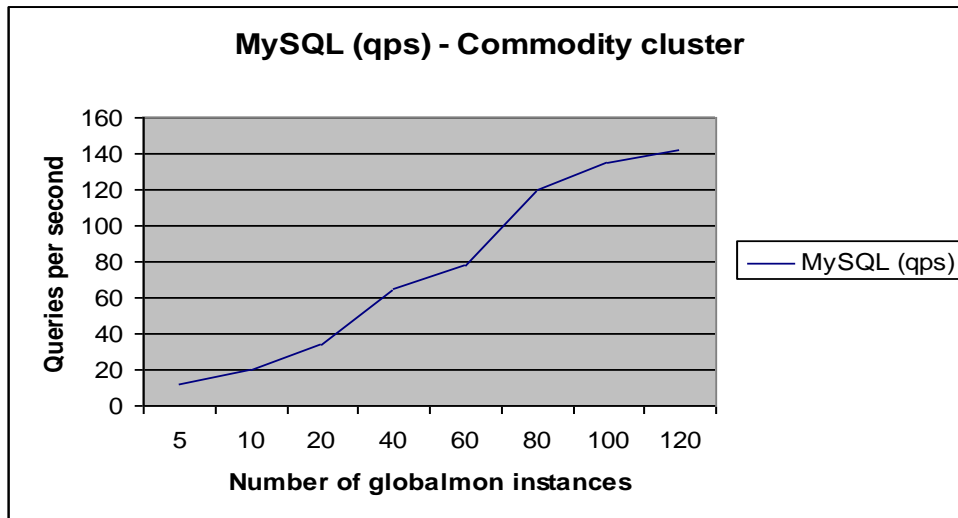


Figure 7.9: MySQL server performance – commodity cluster approach

7.5 Conclusions

This chapter presented the details of the ITPM prototype system architecture. The development of an early prototype system was necessary, in order to refine the proposed model and lay the foundations for further research and development efforts in the field of Insider Threat Misuse Prediction. The design addressed the originally envisaged goals for a scalable, secure and Operating System neutral prototype system, in order to facilitate experiments on Insider Threat Prediction.

CHAPTER 8 CONCLUSIONS

The previous chapters of the thesis have presented the problem of Insider Misuse and discussed how it is possible to devise a systems architecture that predicts the occurrence of insider threats in a computing infrastructure. This chapter offers a critique on the perceived achievements and limitations of the research project, with regards to its initial objectives stated in section 1.1 of the thesis. The first section evaluates the accuracy of the insider misuse research field investigation that the research project has performed. Section 8.2 criticizes the core deliverables of the thesis. The strengths and weaknesses of the ITPM system are discussed. The chapter concludes with a suggestion of additional methods to address the perceived limitations of this architecture and guide future research efforts in the field.

8.1 On the accuracy of the Insider Misuse survey and the preliminary analysis of insider threats

The first objective of the research project was to provide a total overview of the Insider Misuse problem. Chapters 3 and 4 attempted to address this issue. The earlier presented an overview of the problem, mainly by analyzing data from the CSI/FBI survey and the consideration of real-world cases. The latter chapter devised a bespoke survey for the Insider Misuse problem as part of this research project, based on a small sample of fifty computer professionals.

The 2001 CSI/FBI Computer Crime and Security survey [55], has the following comment by Schultz: *“I would like to add that any statistics concerning security related incidents should not be taken at face value...”*. Moreover, the DTI/PWC 2004 survey mentions that: *“... They [surveys] also tend to be biased towards larger and more security-aware organizations...”*. Both of these statements indicate that the goal of an information security survey is to reveal broad incident trends, not make warranties about absolute numbers validated by accurate statistics.

However, [52] suggested an informal way to verify the validity of a survey, which is not other than a comparison of the survey with other similarly minded surveys. Since this research project contributed the small scale insider misuse survey, the validity of this contribution could be cross-checked by

comparing the derived conclusions with the 2003 CSI/FBI Computer Crime and Security survey [53], as well as the most recent DTI/PWC 2004 survey [52]. Section 4.6 discussed the similarities of the project's Insider Misuse survey to the 2003 CSI/FBI Computer Crime and Security survey. The following paragraphs will examine notable similarities between the data derived from the Insider Misuse survey and the DTI/PWC survey. Some aspects of the conclusions derived in chapter 3 are also in line with the results of the DTI/PWC survey and they are also considered.

From a statistics point of view, nearly a quarter of the DTI/PWC participants have stated that their worst security incidents have originated by internal user actions. The difference between this figure and the one quoted by the Insider Misuse survey (70%) is a large one, indicating the bias of the Insider Misuse survey towards Insider Misuse incidents. On the other hand, the DTI/PWC survey mentions that Insider Misuse has doubled since the year 2002, mainly driven by the increased adoption of World Wide Web and Internet related technologies. Consequently, one can repeat the conclusion made in section 4.6: All three surveys indicate that the Insider Misuse problem is a serious threat for the health of IT infrastructures.

In addition, the Insider Misuse survey highlighted roughly the same common types of legitimate user misuse with the DTI/PWC survey. Chapter 4 mentioned that the three most serious (in terms of frequency and disciplinary actions taken as a result of the incident) types of misuse was the downloading of pornographic material, the theft or malicious alteration of data and the abuse of email resources (figure 4.12). In direct comparison, the DTI/PWC highlights the incidents of web browsing misuse, misuse of email and unauthorized access to systems or data as the major system misuse categories.

It is really difficult to compare classes of incidents amongst different surveys, due to the different scope of the incident categories. The 'Web browsing misuse' category of the DTI/PWC survey can include the downloading of pornographic material and other unauthorized use of the World Wide Web facility, mainly for non work related purposes, a scope of misuse that has highlighted by the Insider Misuse survey in figure 4.13. The same can be said about the DTI/PWC 'email abuse category'. Although the Insider Misuse survey considered activities such as spamming or the use of email for abuse or

defamatory purposes, the scope of the DTI/PWC email abuse definition was broader including email utilization for personal purposes, resulting in lost productivity. Nevertheless, the two surveys highlighted three of the most common problem areas in slightly different order.

Another notable similarity between the Insider Misuse and the DTI/PWC surveys is the highlight of staff security checks during the recruiting process. Figure 4.17 of the Insider Misuse survey indicated that all of the surveyed professionals indicated some preference towards the existence of certain pre-employment security checks for prospective employees. The DTI/PWC indicated that the majority (66%) of the respondents usually perform some sort of security check during the recruiting stage. The DTI/PWC survey comments that the absence of these security checks from company procedures is clearly a serious omission.

With regards to the preliminary analysis of the Insider Misuse threat nature (chapter 3), the DTI/PWC survey has revealed another noteworthy conclusion that concerns the relation between internal and external security incidents. Section 3.4 of the thesis concluded that mutual exclusion between internal and external incidents is not an adequate way of analyzing internal threats in a computing environment. The DTI/PWC 'information security breaches survey 2004' authors emphasize that, for the first time, the participants could identify incidents that were caused by a combination of internal and external factors. This verifies the fresh line of thinking which links the two different types of incidents, in order to provide a more holistic approach in the process of understanding the real impact of the insider threats.

Consequently, the thesis has provided a comprehensive overview of the insider misuse problem and satisfied the first two of the four objectives of section 1.1.

8.2 On the Insider Threat Prediction Model architecture

The conclusions derived by the analysis of the various surveys and the discussion of well known insider misuse cases have been used to form a taxonomy of insider threat prediction events (chapter 5). The establishment of the taxonomy was necessary in order to profile the consequences of insider misuse actions at system level. Whilst chapter 5 discussed various types of taxonomies, it introduced

the notion of structuring a taxonomy around system level actions. This idea represents a fresh approach in and created a number of publications ([65],[70],[131]) and formed the basis for a revised Insider Threat prediction taxonomy and the associated ITPM system architecture.

It is really difficult to criticize the effectiveness of the overall architecture for two main reasons. The first one revolves around the fact that, at the time of writing, there were no similarly minded published architectural descriptions and associated system implementation attempts. The insider threat frameworks suggested by Wood [82] and Schultz [83] are preliminary plans and they do not constitute complete architectural attempts that mitigate the problem of insider threat. Whilst this fact indicates the unique contribution of the thesis to the research domain, it also represents an obstacle in the evaluation of the work. One could objectively locate the strengths and weaknesses of the ITPM architecture more easily if there were similar frameworks to benchmark against.

The second –and most important reason- is the lack of insider misuse case data. This research project was based on the systematic examination of legitimate user actions at system level. Although most of these actions could be reproduced by reading about an insider misuse incident and then reproducing the misuse procedure in a simulated environment, this approach might not be the best one for validating the effectiveness of the ITPM architecture. Information security surveys and mass media might report accurately the outline of the case, however they do not provide a complete picture about the conditions under which the incident occurs nor they always reveal fully the timing, commands and the order in which they occur. The lack of much needed insider misuse case repositories is mentioned in [128].

As a result, the critique presented in the following paragraphs is derived in terms of the design philosophy of the proposed architecture and not by real world benchmarks or extensive testing of the proposed architecture.

From an architectural point of view, the ITPM system has combined misuse detection (mostly for the detection of command execution, network and file operations) as well as elements of anomaly detection associated mainly with the process of evaluating the level of legitimate user sophistication. The combination of these two techniques is often encountered in IDS designs, as discussed in section 2.4 of

the thesis. The choice of applying misuse detection to the monitoring of file, network and command execution events was made for the purposes of computing efficiency and also due to the fact that misuse detection is applicable to events that are predictable. Misuse threats are analyzed and the threat signs are known.

Consequently, in the domain of insider misuse threat prediction, misuse detection has the advantage of computational efficiency. The only perceivable disadvantage relates to the fact that the threat prediction success is dependent on the crafting of the misuse signature. If the understanding of the misuse incident is flawed or incomplete, the accuracy of the system will be also flawed. Clearly, the ITPM architecture emphasizes the role of the security specialist, in direct contrast to turnkey solutions that offer heuristics driven by machine learning methodologies.

Despite the aforementioned disadvantage of the ITPM misuse detection components, it should be noted that the philosophy of employing human-driven misuse detection to mitigate security incidents has also been in the core of the computer anti-virus industry with great success. Today, anti-virus products might employ anomaly detection or heuristic-based approaches in the process of detecting and intercepting the actions of malicious computer code. 'Kaspersky Labs' [129], Symantec [130] represent characteristic anti-virus vendor examples that try to enhance the performance of their products by employing heuristics. Nevertheless, the core of their engines is based on misuse detection methods whose signatures are written by computer virus researchers.

A secondary side effect of the misuse prediction parts of the model is related to the complexity of devising the misuse prediction signatures. The plethora of potential scenarios and their subsequent interpretation into suitable file, network and command execution statements can be a daunting task, even for experienced security specialists and busy system administrators. Although the prototype system provided signature creation tools that create the complete multi-level misuse signature (section 6.4), the information is still entered in a mechanistic way.

It would be easier if the information was entered into the system in a more user friendly format, by means of worded statements that conform to an insider misuse prediction specification language. A

language parser could then convert the worded statements into suitable file, network and command execution statements that form the misuse signature. Although the proposed ITPM architecture provides the backbone for the formation of such a language, it focused more on the content of the signatures at system level, without providing a semantic framework for abstracting the misuse statements. Consequently, the derivation of such a language would be a worthy addition to the architecture that would greatly enhance the operational effectiveness of the ITPM system and section 8.3 will propose ways to embed this functionality to the project.

Whilst misuse detection achieves good results when applied to predictable data sets, it is difficult to apply patterns when estimating less certain data. Evaluating user sophistication by means of examining a plethora of computer applications amongst different computing environments requires a different detection approach. As a result, the part of the ITPM model that evaluates user sophistication is based on anomaly detection. The method and its results were submitted by Magklaras and Furnell [131]. It performed really well on the experiment of section 6.3.2.1 and no instances of user misclassification were observed. However, the fact that the model has to be re-trained even after small changes (removals or additions of software applications) to the computing environment reduces the flexibility of this method. This constitutes a serious disadvantage in rapidly changing IT environments.

Nevertheless, the user sophistication component of the ITPM model represents a novel experimental approach that could not only provide a metric for an Insider Threat Prediction process, but which could also be useful for people concerned with the automatic customization of Human Computer Interaction (HCI) interfaces, or people that would like to estimate the productivity potential of their computing users.

Thus, a detailed Insider Threat Prediction model was proposed with its associated architecture for a proof-of-concept system implementation. This satisfies objectives 3 and 4 of section 1.1. However, it becomes clear that the proposed system requires more rigorous validation. A greater number of incidents must be tested in the model, under different types of computer environments. Unfortunately, the lack of a central insider misuse case data repository and the time scales of this project did not leave

room for performing a more systematic evaluation of the model. The next section will discuss future research directions that will make the process of validating the model feasible and more effective.

8.3 The research continues

Whilst the ITPM architecture presents a first step towards the construction of tools that mitigate the insider threat, it is by no means a complete design. There are several challenges that the research project has not addressed due to time limitations as well as problems that were perceived after the completion of the experiments. The addressing of these issues will perfect the ITPM architecture and guide the production of more accurate research and development insider threat prediction tools.

8.3.1 The insider misuse case data repository

Section 8.2 mentioned the lack of insider misuse case data repositories. A central database of insider misuse cases built around the ‘signatures’ table (Figure 7.2) of the ITPM database is a very important step that would be of great aid to insider misuse researchers worldwide. The maintenance of the database should be done by experts that collect the data from real-world computer crime scenes.

The field of computer forensics offers methodologies that perform the necessary data collection procedures in a standard manner. Vacca [132] defines computer forensics as “...the collection, preservation, analysis and presentation of computer related evidence”. Thus, standard computer forensics software could be employed to recover the necessary data and then further tools need to be created in order to construct the necessary misuse signatures for the ITPM architecture and populate database repositories. If a researcher wishes to address a particular problem, he could then browse the database for incident-driven recipes of misuse signatures and then modify them appropriately, in order to construct misuse threat signatures tailored to his environment.

The existence of such a data repository could also decrease the signature construction time and act as a benchmark for comparing different methodologies or models and tools for mitigating insider threat. If one has access to accurate and standard ways of reconstructing real-world cases, he could more easily establish testing frameworks for assessing the prediction accuracy of different models.

8.3.2 Towards an Insider Misuse Threat Prediction Specification Language

The critique of section 8.2 also referred to the complexity issues that surround the construction of misuse signatures in the proposed ITPM architecture and talked about the construction of a complete insider misuse threat prediction specification language. Such a language would be a special case of a Domain Specific Language (DSL), a semantic mechanism tailored specifically for describing the details of a particular task [133]. The main goal is the usage of appropriate semantics to reduce the effort required to reference and manipulate elements of that particular domain. Thus, a methodology for deriving a Domain Specific Language includes three important steps:

- The abstraction of the domain, which involves the removal of all the unnecessary details of the environment.
- The systematic categorisation of the necessary (abstracted) details into language semantics.
- The process of engineering the developed semantics into software.

The proposed Insider Misuse Threat Prediction event taxonomy (chapter 5) as well as the derived Insider Threat Prediction model represent the abstraction of the problem domain. The proposed misuse signature encoding semantics could form parts of such a language but they lack the completed semantic framework. The next paragraphs will represent some notable research and development efforts that could be re-used to form the Insider Misuse Specification language.

Section 2.5 of the thesis has mentioned the troubled Common Intrusion Detection Framework (CIDF) [29], whose scope of work has been taken over by the IETF Intrusion Detection Message Exchange Format working group [30]. The framework's Common Intrusion Specification Language (CISL) [134] consists of a semantic framework to unambiguously describe intrusive activities together with proposed data structures that store the event information and can form standardised messages exchanged by various IDS components.

The CISL framework could be re-used for producing a suitable Insider Misuse Threat Specification Language. However, the framework would require substantial re-engineering. The following

paragraphs discuss the CISL framework the latest research efforts associated with it, present the major flaws it has and suggest a research and development methodology to eliminate these problems.

In CISL, the semantic representation of intrusive activities is achieved by the formation of an S-Expression. An S-Expression is a recursive grouping of tags and data, delimited by parentheses. The tags provide semantic clues to the interpretation of the S-Expression and the data might represent system entities or attributes. For this reason, the tags are also called Semantic Identifiers (SIDs).

The best of way of illustrating how CISL works is by considering an example. The statement (*Hostname 'frigg.uio.no'*) is a simple S-Expression. It groups two terms, without semantically binding them. One can guess that it refers to a computer system with the FQDN name 'frigg.uio.no', but the true meaning of the statement is still vague. In fact, the full semantic meaning of S-Expressions becomes apparent when one forms more complex S-Expressions, by means of combining several SIDs into a sentence. Figure 8.1 provides an example of a CISL sentence.

```
(And
  (OpenApplicationSession
    (When
      (Time 14:57:36 24 Feb 2004)
    )
    (Initiator
      (HostName 'outside.firewall.com')
    )
    (Account
      (UserName 'tom')
      (RealName 'Tom Attacker')
      (HostName 'frigg.uio.no')
      (ReferAs 0x12345678)
    )
    (Receiver
      (StandardTCPPort 22)
    )
  )
  (Delete
    (World Unix)
    (When
      (Time 14:58:12 24 Feb 2004)
    )
    (Initiator
      (ReferTo 0x12345678)
    )
    (FileSource
      (HostName 'frigg.uio.no')
      (FullFileName '/etc/passwd')
    )
  )
  (OpenApplicationSession
    (World Unix)
```

```

    (Outcome
      (CIDFReturnCode failed)
      (Comment '/etc/passwd missing')
    )
  (When
    (Time 15:02:48 24 Feb 2004)
  )
  (Initiator
    (HostName 'hostb.uib.no')
  )
  (Account
    (UserName 'ksimpson')
    (RealName 'Karen Simpson')
    (HostName 'frigg.uio.no')
  )
  (Receiver
    (StandardTCPPort 22)
  )
)
)
)

```

Figure 8.1: CISL sentence syntax example

The CISL sentence of Figure 8.1 could be translated in the following plain English translation:

“On the 24th of February 2004, three actions took place in sequence in the host ‘frigg.uio.no’. First, someone logged into the account named ‘tom’ (real name ‘Tom Attacker’) from a host with FQDN ‘outside.firewall.com’. Then, about a half-minute later, this same person deleted the file ‘/etc/passwd’ of the host. Finally, about four-and-a-half minutes later, a user attempted but failed to log in to the account ‘ksimpson’ at ‘frigg.uio.no’. The attempted login was initiated by a user at ‘hostb.uib.no’.”

The particular CISL sentence describes a malicious attack that erases an important system file of a UNIX system and consists of three multi-SID S-Expressions. In general, a sentence can be formed by one or more S-Expressions nested at different levels. However, there are strict rules that allow the nesting of S-Expressions. The rules are defined by the nature of the SIDs, as there are several different types of them.

Every CISL sentence must contain at least one verb SID (in the example ‘Delete’), denoting some sort of action or recommendation. Verb SID’s are joined together in a sentence by conjunction SIDs. In the previous example ‘And’ is the conjunction SID that holds together the three SIDs that form the sentence. In addition, a CISL sentence might employ role, adverb, attribute, referent and atom SID

types. There are additional SID types but the aforementioned ones are the most commonly employed ones.

Role SIDs indicate what part an entity plays in a sentence (such as 'Initiator'). Adverb SIDs provide the space and time context of a verb SID. Attribute SIDs indicate special properties or relations amongst the sentence entities, whereas atom SIDs specialise in defining values that are bound to certain event instances (for instance 'Username'). Lastly, referent SIDs allow the linking of two or more part of a sentence ('Refer to' and 'Refer as').

The wealth of SID types increases the semantic expressiveness of the language, but there is also a structural hierarchy for forming complex sentences that also contributes to the semantic meaning. This semantic structure is similar to the syntax of natural languages. A verb SID is always at the heart of every CISL sentence and is followed by a sequence of one or more S-expressions that describe the various entities that play parts in the sentence, or qualify the verb. As a result, under a verb SID one can find nested S-expressions headed by a role SID. Under the role SID, one can find atom and adverb SIDs.

This hierarchy is rigid and forms part of the CISL language. A similar hierarchy can be observed in the formation of file and network level ITPM expressions in chapter 6.

The second part of the CISL language specification [134] is concerned with the encapsulation of the structured semantic information into the Generalised Intrusion Detection Object (GIDO). GIDOs are data structures that hold the encoded event information. The purpose of encoding the information in a standard way is to make the process of exchanging the information amongst various CIDF components easy, in order cross-vendor IDS interoperability recipes.

Unfortunately, despite the well-conceived interoperability target, the CISL GIDO encoding process introduced many problems. Doyle [135] has criticized many of the aspects of the CISL GIDO structure. Although the purpose of the document was to evaluate the fitness of CISL for use in the DARPA Cyber Command and Control (CC2) initiative, the paper locates serious inadequacies that concern the CISL

time resolution data representation facilities, as well as data throughput limitations caused by the fixed size of the GIDO data structure. Finally, Doyle comments on the lack of support for the next generation Internet Protocol (Version 6). Whilst these points are fair, they could easily be corrected by making the necessary changes to the relevant data types and overcome the perceived obstacles. In fact, section 7 of the CISL standard [134] contains specific guidelines that explain how to add information to a GIDO, to clarify or correct its contents. This suggests that the encoding principles are certainly extensible.

A more serious aspect of Doyle's critique [135] refers to the semantic structure of the CISL language. In particular, his criticism that CISL has "no facilities for representing trends or other complex behavioural patterns; ill-specified, inexpressive, and essentially meaningless facilities for representing decision-theoretic information about probabilities and utilities" indicates that the language would be a bad choice for describing information about a threat prediction model. The basic reasoning behind this critique is that CISL is too report-orientated and threat mitigation requires a different level of information, not just mere report structures of what is happening on one or more systems. These indeed represent more serious limitations that would require a more radical re-design of the CISL.

In response to the CISL encoding limitations, the IETF Intrusion Detection Exchange Format working group [30] took over the scope of the CIDF work. It addressed most of the GIDO encoding issues by introducing a new Object Oriented format for encoding and transmitting Intrusion Detection related information. The Intrusion Detection Message Exchange Format (IDMEF) [136] enriched the type of standardized information that IDS sensors may represent, as well as the process of standardizing the exchange of messages using protocols such as IDXP [137] and data exchange languages such as XML [138].

For example, the IDMEF 'Confidence' and 'Impact' classes can now be used to represent decision theoretic information [136]. The earlier can assign a confidence and thus a probability to an observed event, whereas the latter relates privilege escalation consequences to three broad severity levels. This functionality can server as the basis for encoding probabilistic information, in order to use it in a threat prediction model such as the ITPM.

These standardization features were lacking from the previous CIDF platform and they constitute a very important step towards establishing better interoperability amongst different IDS products. However, at the time of writing, the working group has not managed to correct and standardize the semantic structure of the CISL language. The IDMEF draft standard [135] proposes encoding and data structures, but it does not suggest semantic guidelines like the ones proposed by the CIDF framework. For IDMEF, the term ‘language’ refers to the data types and encoding principles for IDS data and not to the syntactical guidelines of an Intrusion Specification Language.

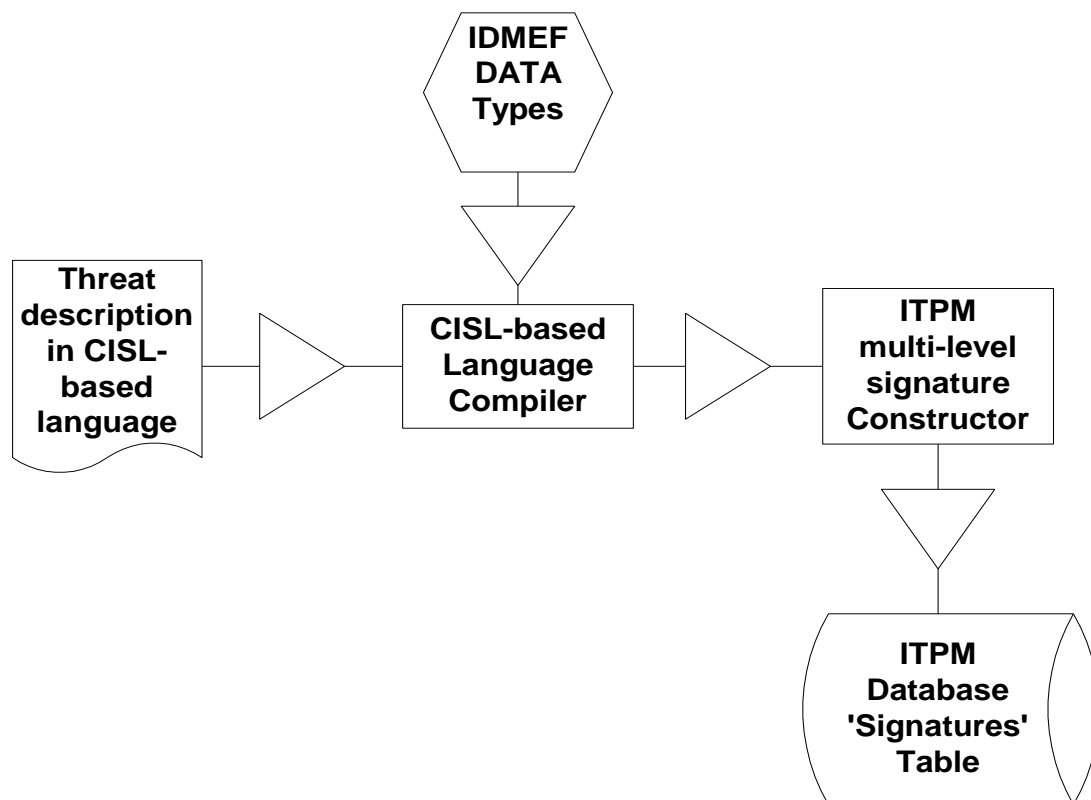


Figure 8.2: Insider Threat Prediction Specification Language data flow

Hence, if one wishes to establish an Intrusion Specification Language tailored to Insider Threat Prediction, he would have to adopt the basic syntactic guidelines of the CISL and address the syntactic inadequacies indicated by Doyle [134]. After the semantic refinement step, an effort to match the suggested ITPM expression data to the IDMEF data structures should take place. This will ensure that the ITPM architecture would be fully compliant with the relevant standards of the research field, in order to be interoperable with many IDS products. The last step would be to write the language compilers and link them to the signature construction tools. Figure 8.2 illustrates the process of turning a CISL-based plain text description into a multi-level threat prediction signature (section 6.4 of the thesis).

The process of refining the original CISL semantic schema would enrich the original language by adding new atom and adverb SID types that represent decision-theoretic and probabilistic information. These new SID types would relate network, file and command execution to weight matrix options, so that a full multi-level threat prediction signature could be constructed.

The process of matching the ITPM proposed data to the IDMEF event classes should also not pose a major problem. Weight Matrix data could be represented by the confidence attributes of the aforementioned IDMEF 'Assessment' class. User privileges can be represented by the 'Impact' class. In addition, there are plenty of IDMEF data structures that can represent information related to the file, network and command execution ITPM components. The 'FileList' and 'FileAccess' classes contain adequate attributes to represent the file attributes. The 'Address' class can represent network related data, and lastly, the 'Process' class could accommodate most of the requirements of the command execution data of the ITPM architecture.

The fact that the IDMEF draft standard has similar data type mappings with the proposed ITPM architecture indicates that the research project has moved on the correct track when it comes to IDS interoperability. However, the process of constructing complete semantic frameworks and performing a correct one-to-one mapping between the ITPM and IDMEF data types is a formidable task.

In addition, one would have to write the complete language compilers and prove the effectiveness of the deduced language on real-world scenarios. For these reasons and although this research project has emphasized the importance of deducing such a language, it could not accommodate the production of the Insider Threat Prediction Specification Language within the available time scales. Nevertheless, it provided a strong foundation for the building of the language by abstracting the problem domain and suggesting formats for the encoding of the threat prediction signatures.

8.4 Epilogue

This final chapter has provided a critique of the overall project by discussing the accomplishments as well as the limitations of the produced ITPM architecture. In essence, the goal of the thesis was to shed light to the problem of insider misuse in IT systems and propose novel ways to estimate forthcoming insider threats. The goal of these threat estimation techniques was not to provide a panacea against all the malicious or accidental actions of legitimate users but a complement to existing security monitoring tools.

Whilst the thesis has managed to offer a comprehensive picture of the nature and the magnitude of the legitimate user misuse, the proposed novel ITPM architecture needs further research and development efforts prior proving itself as a reliable, production-grade system that mitigates the problem of insider threats. The lack of formal insider misuse case data repositories limits the necessary validation and testing efforts of the devised model. In addition, the perceived complexity of the model necessitates the development of an Insider Threat Prediction Specification Language, in order to increase the flexibility and compliance of the model with the new standards of the IETF Intrusion Detection Message Exchange Format Working Group [30].

Nevertheless, the proposed ITPM architecture with all of its novelties and imperfections presents a step forward in the Insider Misuse Threat mitigation field and will hopefully inspire researchers to improve and complete their own frameworks that address the same problem.

Appendices

Appendix A: Commercial IDS vendors

The main body of the thesis discusses many IDS efforts that originated mainly from an academic research and development environment. Although it is outside the scope of this thesis to provide a detailed overview of the functionality of commercial products, IDS vendors have presented their own solutions. It is worth devoting an Appendix to look at the various IDS paradigms they introduce and see their advantages and disadvantages in relation to existing academic efforts.

It should be noted that the contents of the following paragraphs do not address the wealth of existing commercial products for two reasons:

- It is impossible to review every single product of the large and rapidly expanding IDS market.
- The cost of some commercial products and the 'closed source' model made it impossible to investigate their functionality in a detailed manner.

With these factors in mind, the following paragraphs present the generic philosophy of commercial IDS designs.

Most well-known IDS products follow employ both of the major IDS detection engines described in Chapter 2 of the thesis. Chapter 2 outlined the strengths and weaknesses of Misuse and Anomaly Detection methodologies. Commercial IDS Vendors combine these techniques in their effort to reduce the Detection efficiency of their products. This is an idea that was originally introduced in the Academia, with the development of the first Intrusion Detection System Frameworks.

Hence, it could be argued that from an algorithmic novelty point of view, commercial IDS systems present no radically new elements. Their novelty lies mainly in the area of improving these algorithms in terms of computational efficiency.

SNORT [34] is an Open Source Intrusion Detection System that is however sponsored and partly developed by 'SourceFire Incorporation', a commercial company that specialises in Intrusion Detection. It uses misuse detection to analyse network and (in some modes) host related data. The

latest version of the product (Version 2.0) utilises a misuse detection engine that is based on Wu and Manber's work on a Fast Pattern Matching Algorithm [35]. The algorithm was developed in academia and it increases the effectiveness of misuse detection by performing multiple searches on pre-selected intrusion signatures. The end result is that the runtime of the detection process is substantially reduced by utilising a combination of light-weight processes and memory management techniques.

The novelty of the SNORT product is the application of this algorithm to its own proprietary Rule Definition Language, in order to increase the effectiveness of the misuse detection process [35]. Wu and Manber's algorithm was established with reference to the generic pattern matching domain and it was not developed with IDS designs in mind. However, 'SourceFire' developers have constructed their own framework, in order to apply this algorithm to the IDS problem domain. This framework marketed with the name '*Real Time Network Awareness*TM' consists of the SNORT Rule Definition Language, a bespoke implementation of the misuse detection engine and often pre-configured hardware systems that run the IDS engine on a highly customised Operating System [37].

The approach of combining customised hardware and an Operating System to run IDS software is another distinct trend of commercial products, creating 'turnkey solutions'. These types of solutions are favourable by the commercial world, as they require less specialised resources to deploy them and they are more easily manageable than software solutions. '(nfr) (security)' [38] is a widely respected Network IDS vendor that delivers the 'NID' family of turnkey solutions. These systems consist of a client-server system that is accompanied by sensor boxes. The client is an administration Graphical User Interface (GUI) that normally runs on a Windows -based workstation. The server houses the 'turnkey' box and the sensors consist of software components installed in various places of an IT infrastructure, as figure A1 illustrates.

Another distinct trend of commercial IDS solutions is the continuous integration of IDS and network management systems (NMS), a feature that is not addressed in academic designs. An NMS is a software solution that allows IT specialists to configure, troubleshoot and objectively characterise the performance of a computing infrastructure in an efficient manner [39]. The Simple Network Management Protocol (SNMP) Version 3 [40] as well as the Remote Monitoring (RMON) Management Information Base [41] are two standardised protocols that are employed to achieve the

NMS/IDS integration. As a result, the NMS functions of configuration, fault and performance management embed IDS functionality and use the wealth of information provided by NMS sensors to feed IDS engines with valuable data for the status of the IT infrastructure.

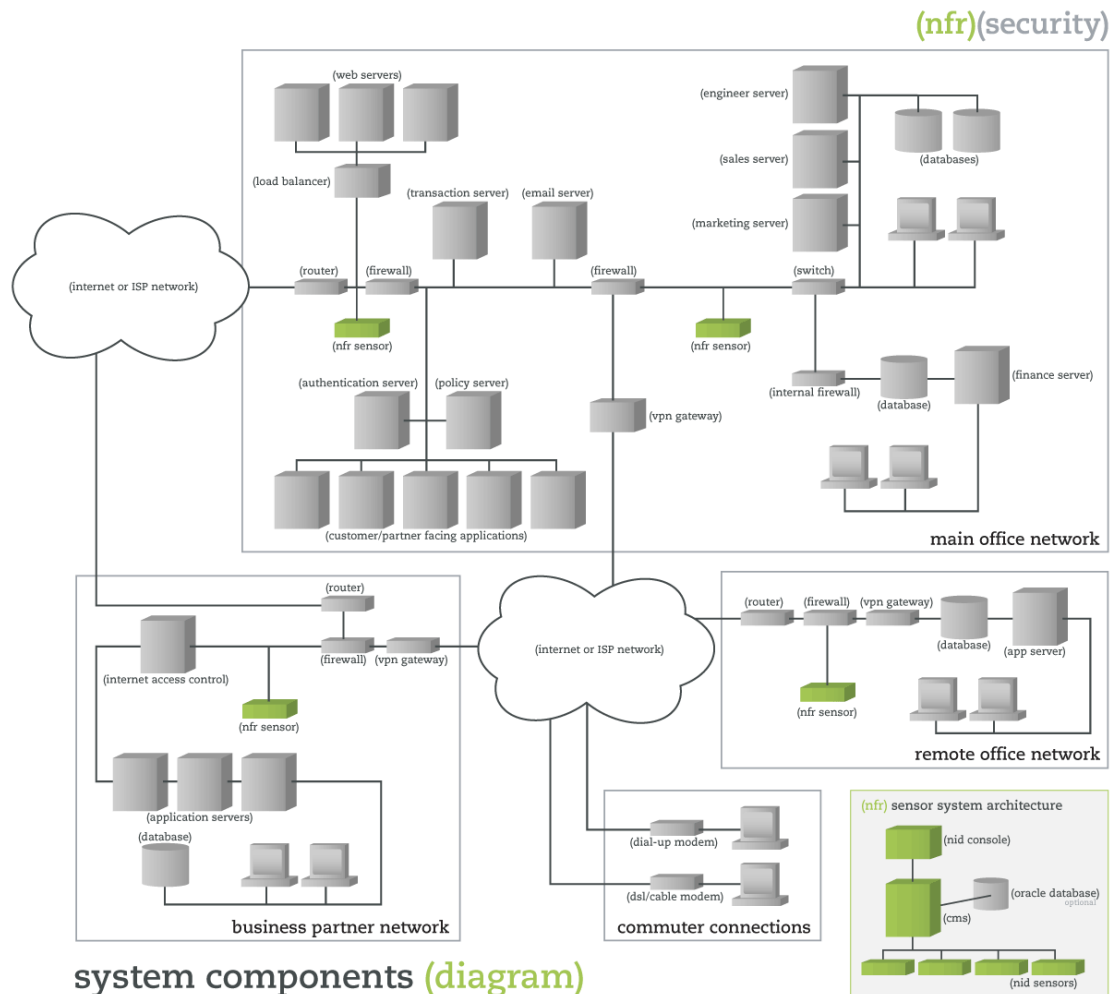


Figure A1: NFR Security IDS architecture

The ‘RealSecure’ IDS engine from Internet Security Systems (ISS) [42] was one of the first examples that addressed the issue of NMS/IDS integration. It can utilise the well-known Hewlett Packard’s ‘Open-View’ NMS engine [43] as an underlying mechanism for collecting real-time data from several components of the IT infrastructure. Other IDS vendors follow similarly layered approaches in their design of their products.

Whilst the majority of commercial IDS designs tend to address mainly the problem of externally initiated attacks, the same cannot be said about internal attacks [44]. Clearly, features such as internal activity tracking represent a new trend in commercial IDS products. The ‘e-trust’ software suite by ‘Computer Associates’ [45] is a classic example of a comprehensive IDS tool, which analyses a series

of system log files and helps system administrators pinpoint employee activities. The monitored activities include e-mail and web page visits for each individual user. The 'Fire-Marshall' IDS series of 'Palisade Systems' [46] is also another case of Insider activity tracking tools. Apart from the usual employee web and e-mail monitoring facilities, the 'Fire-Marshall' product includes the ability to measure network bandwidth consumption violations and block certain access attempts via certain network endpoints, providing a comprehensive engine for the detection of certain internal activities.

Critique of commercial IDS offerings

There is no doubt that commercial vendors have contributed towards the acceptance of the IDS concept in the IT world by improving the performance of the IDS engines as well as making products that can be easily deployed and maintained. However, the fundamental IDS design challenges are still not addressed.

Commercial IDS solutions still suffer from false negative/positive alarms, another indication of their non-existing algorithmic novelty. Consequently, there is still plenty of room for research and development on improving the efficiency of anomaly and misuse detection algorithms.

The usability, performance enhancement and NMS integration features are very useful in developing real-world applications. However, they have created a number of other problems that affect interoperability functions amongst different IDS products.

Modern IT infrastructures are non homogeneous. An IT security architect often has to choose amongst many products to provide the optimum solution for a number of Data Security problems. Whenever these products are not able to exchange data efficiently, the entire process becomes a burden and the architect is often locked to expensive and inefficient single vendor solutions.

This is a very familiar picture in the commercial IDS world. For the great majority of the products, it is impossible to exchange IDS-related data. Intrusion signatures and anomaly detection rules are vendor-specific. Normally, they cannot be shared amongst different products. Some vendors have recently considered the exception to this rule and they made their products recognise rules and signatures from

other vendors. However, currently this is considered as an add-on feature rather than standard functionality.

An additional negative point of some IDS vendors is the level of automation they introduce in the process of configuring and deploying their products. The previous section gave a brief overview of 'turnkey' solutions. These solutions are marketing the easy deployment and maintenance features and their vendors always emphasize the minimum level of training required, in order to operate the product. However, the policy of not maximizing the training of IT staff on security or hiring Data Security specialists to protect your IT assets for the purposes of reducing the operating costs is a dangerous practice. Effective Data Security cannot be achieved by the deployment of usable and automated products that you place in your network and you forget about them. An IT infrastructure always needs knowledgeable specialists that can further customise these products to suit the needs of their organisation.

For all these reasons, despite the existence of a vast market of IDS turnkey product solutions, research and development issues that concern key IDS algorithmic concepts is still in a state of flux, leaving plenty of room for further research and development efforts.

APPENDIX B: CSI/FBI 2003 Computer Crime Survey

The Cost of Computer Crime													In 2003, 75% of our survey respondents acknowledged financial losses, but only 47% could quantify the losses.				
The following table shows the aggregate cost of computer crimes and security breaches over a 48-month period																	
How Money Was Lost																	
	Lowest Reported				Highest Reported				Average Losses				Total Annual Losses				
	00	01	02	03	00	01	02	03	00	01	02	03	00	01	02	03	
Theft of proprietary info.	\$1K	\$100	\$1K	\$2K	\$25M	\$50M	\$50M	\$35M	\$3,032,818	\$4,447,000	\$6,571,000	\$2,699,842	\$66,708,000	\$151,230,100	\$170,827,000	70,195,000	
Sabotage of data of networks	1K	100	1K	500	15M	3M	10M	2M	969,577	199,350	541,000	214,521	27,148,000	5,183,100	15,134,000	5,148,500	
Telecom eavesdropping	200	1K	5K	1K	500K	500K	5M	50K	66,080	55,375	1,205,000	15,200	991,200	886,000	346,000	76,000	
System penetration by outsider	1K	100	1K	100	5M	10M	5M	1M	244,965	453,967	226,000	56,212	7,104,000	19,066,600	13,055,000	2,754,400	
Insider abuse of Net access	240	100	1K	100	15M	10M	10M	6M	307,524	357,160	536,000	135,255	27,984,740	35,001,650	50,099,000	11,767,200	
Financial fraud	500	500	1K	1K	21M	40M	50M	4M	1,646,941	4,420,738	4,632,000	328,594	55,996,000	92,935,500	115,753,000	10,186,400	
Denial of service	1K	100	1K	500	5M	2M	50M	60M	108,717	122,389	297,000	1,427,028	8,247,500	4,283,600	18,370,500	65,643,300	
Virus	100	100	1K	40	10M	20M	9M	6M	180,092	243,835	283,000	199,871	29,171,700	45,288,150	49,979,000	27,382,340	
Unauthorized insider access	1K	1K	2K	100	20M	5M	1.5M	100K	1,124,725	275,636	300,00	31,254	22,554,500	6,064,000	4,503,000	406,300	
Telecom fraud	1K	500	1K	100	3M	8M	100K	250K	212,000	502,278	22,000	50,107	4,028,000	9,041,000	6,015,00	701,500	
Active wiretapping	5M	0	0	5K	5M	0	0	700K	5M	0	0	352,500	5,000,000	0	0	705,000	
Laptop theft	500	1K	1K	2400	1.2M	2M	5M	2M	58,794	61,881	89,000	47,107	10,404,300	8,849,000	11,766,500	6,830,500	
CSI/FBI 2003 Computer Crime and Security Survey Source: Computer Security Institute												Total Annual Losses		265,337,990	377,828,700	455,848,000	201,797,340

Table B1: The cost of Computer Crime according to the 2003 CSI/FBI Survey

Appendix C: Insider IT Misuse Survey Questionnaire

Identification information:

You don't have to complete the following five fields. However, if you wish to do so, please make sure that you fill in at least the 'Name of Organisation/Company' and the 'Address' fields. **Please use the TAB key or the mouse cursor (left click) to move to the next input field.**

Name of Organisation/Company	<input type="text"/>
Name of Employee	<input type="text"/>
Address	<input type="text"/> <input type="text"/>
City	<input type="text"/>
Postal Code	<input type="text"/>
Telephone	<input type="text"/>
Email	<input type="text"/>
Country	<input type="text"/>

Part A

1) What is your role inside your organisation/company?

- System Developer
- Security Specialist/
Consultant
- System Administrator
- Manager/Director/CEO

2) How many years of experience do you have in this role?

- One Year _____
- Two Years _____
- Three Years _____
- Four Years _____
- More than Five Years _____

3) In which of the following IT sectors does your company/organisation belong?

- Financial _____
- Education _____
- Defense _____
- Manufacturing _____
- Internet Service Provider _____
- Hardware/Software Vendor _____
- Utilities (Electricity/
Water Supplies...) _____
- Government _____
- Transportation _____

4) State the number of desktops in your organisation.

- 1-10 _____
- 20-50 _____
- 50-100 _____
- 100-500 _____
- 500-1000 _____
- 1000-5000 _____
- 5000 + _____

5) Does your organisation employ a 'firewall' and/or antivirus and/or data encryption product?

- Yes, we use these technologies _____
- Yes, but they are not very effective _____

- No, but we are thinking of installing them _____
- No, and we believe we do not need them _____

6) Does your organisation employ an Intrusion Detection System?

- Yes, we use IDS technology _____
- Yes, but it is not very effective _____
- No, but we are thinking of installing it _____
- No, and we believe we do not need it _____

7) Which of the following Operating Systems do you employ in your business/organisation?

- Microsoft Windows NT 4 Server/Workstation _____
- Microsoft Windows 9x (95,98, 98 SE/ME) _____
- Microsoft Windows 2000 _____
- Novell Netware _____
- UNIX-like (HP-UX, SOLARIS, AIX, LINUX...) _____
- Other (please specify) _____

Part B

8) How many IT security incidents did you approximately have since January 2001?

- 1-5 _____
- 5-10 _____
- 10-20 _____
- 20 + _____

9) Would you say that the great majority of the incidents were due to actions from:

- An unknown origin (I don't know if the Origin was internal or external) _____
- Legitimate users of your organisation _____
- Unauthorised users outside your organization _____

10) In case you have experienced 'insider' incidents, would you say that most of the incidents were the result of:

- An accident _____
- An intentional action (no accident) _____
- I don't know if it was intentional or accidental _____
- Not applicable _____

11) Did an identified 'insider' incident result in a substantial financial loss for your company/organisation? You might also like to include in this answer the cost of any legal proceedings (see questions 12 and 13).

- We lost a considerable amount of revenue _____
- We have not lost a considerable amount of revenue _____
- We have not lost any money as a result of the the incident _____
- Not applicable _____

Estimated Lost Revenue (Optional) _____

12) Did an identified 'insider' incident result in legal prosecution of your company organisation, as a result of an action related to the incident?

- Yes, unfortunately, we were prosecuted. _____
- No, we were not prosecuted _____

13) In cases you successfully identified employees misusing IT resources, did you think it was necessary to prosecute them?

- Yes, we did prosecute employees. _____
- No, we did not prosecute employees _____

Part C

14) Which of the following actions would you your IT security policy consider as insider misuse (Please tick all that apply)?

- Attempt to use an installed application (or range of applications) without authorization ___
- Attempt to install an application (or range of applications) without authorization ___
- Attempt to attach hardware peripherals to desktop systems without authorisation. ___
- Use a particular IT resource (CPU time, network bandwidth..) excessively. ___
- Use the IT resources extensively for purposes other than work (ie Internet Job browsing). ___

15) If you were designing a security pre-employment screening procedure for candidate employees, what would you think is the most important piece of information that should be included in the screening policy?

- Credit difficulties of the candidate employee ___
- Criminal record of his/her family ___
- Level of IT security knowledge ___
- Validation of reasons for leaving previous jobs ___
- None of the above ___

16) Which of the following constitutes the most indicative source for signs of insider misuse incidents?

- Operating System Log files (audit trails) ___
- Application specific log files ___
- Specialised security log files ___
- Social Engineering (chat, rumours, etc) ___
- Information from pre-employment screening procedures ___

- The content of the web pages that the user visits _____
- E-mail content _____
- User generated Network Traffic (type and amount) _____

17) Do you believe that the level of IT knowledge/sophistication of a user could potentially indicate the likelihood of the user abusing the IT infrastructure?

- Yes, sophisticated users are more likely to abuse their systems _____
- No, user IT sophistication is not important _____

18) Based on the experience you gained from the occurred insider incidents, which of the following types of IT misuse incidents do you think that an insider is most likely to initiate?

- Email abuse (spam or abusive defamatory material) _____
- Computer virus implantation _____
- Theft of confidential information _____
- Physical vandalism of It components _____
- Installation of illegal software _____
- Downloading pornographic material from the Internet _____

Appendix D: How a misconfigured system can be susceptible to an insider Denial Of Service attack.

The Incident background:

The presentation of this incident is a result of an external consultancy project that involved the analysis of a commercial LINUX departmental file server. Around 21:00 hours in the evening of the 14th of September 2003, one of the late night users of the machine had noticed that the system started becoming slow. It then refused to provide access to the shared user areas via both the NFS and CIFS/SAMBA services. It would even refuse access from the root super-user which meant that the system was unusable. It would later become obvious that this was the result of a Denial of Service attack. The attack was initiated by a legitimate user of the system, with no escalated privileges in the server. It took the company two working days to restore the system back to a production state. The company name and some of the system details have been omitted from this report.

The data collection and laboratory setup methodology:

After removing the system from the Data Network, a LINUX rescue system disk was created (a modified version of Tom's BRT floppy) to boot the server with a bespoke mini-root filesystem and WITHOUT booting from the original system partitions. The goal was to preserve valuable data from the client as well as evidence for forensic analysis purposes. Thus, the 'dd' image utility was then utilized, in order to make an exact image of the system partitions (including the system's boot sector) to an external USB high-capacity portable hard disk drive. The drive was then taken to the author's own private laboratory for further analysis.

The USB high capacity drive was then used once more with the dd utility to copy the retrieved system partitions and boot sector to a hardware-compatible system with empty hard disks. In that way, the exact content of the partitions retrieved from the client would be preserved without interference induced by the examination procedure.

The forensic analysis of the system:

The system was booted with all the partitions and the boot sector. An attempt to login as 'root' was unsuccessful. The system recognized the super-user account, but the login process would terminate before providing access to the shell prompt. As a result, the rescue disk was used once more to boot the system to a mini-root filesystem.

The 'original' filesystem was then mounted on an alternative mount point, in an attempt to find the reason for the login failure. After issuing a query for the file space capacity of the system, it immediately became evident that the root partition was full:

```
[root@rescuesys root]$ cd /mnt/oldroot; df -h ./
Filesystem      Size  Used Avail Use% Mounted on
/dev/hda1       1012M  961M   0 100% /mnt/oldroot
```

As the /var and /tmp partitions were under the / partition, this meant that the system logging functions (normally under /var/log, the mail spool (usually under /var/spool/mail), as well as other applications that depend on available temporary file space would be unable to operate. This fact was probably a good reason to explain the super-user login failure. Indeed, an examination of the standard user login scripts (in a linux system are normally under /etc and /etc/profile.d directories) revealed that the company had used a bespoke scripting solution that would kill the shell login procedure, if the shell process would not be able to create a file on the user's home directory. In this particular case, the user was root, the home of the root user is under / and the / partition was full. A removal of this rule from the profile scripts and a reboot, allowed the superuser login to complete properly.

Now, the partitioning scheme of the 'live' system looked like this:

```
[root@fileserv01 ~]# df -h
Filesystem      Size      Used      Avail      Use%      Mounted on
/dev/hda1       1012M     1012M      0          100%      /
/dev/hda5       400.0G    399.9G     0          100%      /home
none            90M       0           90M        0%        /dev/shm
/dev/hda2       4.0G     2.9G       892M       77%       /usr
/dev/cdrom      267M     267M       0          100%      /mnt/cdrom
```

Immediately, the fact that the /home partition was full was observed. This obviously explained the failure of other users to login and use (particularly write to) their home areas. The next logical question is obviously ‘which application(s) is/are responsible for this result and who invoked them’.

The first step in answering this question is what were these areas filled up with. A quick look at the / partition by means of using the du utility revealed that the root partition had a lot of data under its /var subdirectory, accounting for 804 of the 1012 Megabytes available to the partition.

```
[root@fileserv01 ~]# du -m -s /var
804  /var
```

The first suspicion was a logging or mail spool process that went out of control, but after examining the directory hierarchy under the /var subdirectory the guilty file was found. Under /var/tmp there was a strangely looking file with a size of 748 Megabytes. The file was just full with null bytes and contained no further information.

```
[root@fileserv01 tmp]# ls -lh
total 749M
-rw-rw-r-- 1 usr547  usr547   748M Sep 14 20:43 jj123-ssh
```

In addition, the owner of the file was user547. After quickly calling the company’s personnel department, it was established that user547 was an applications programmer working as a software development consultant, who had just completed a large software project with the company.

In an attempt to discover what was the real connection of that file and that particular system user, the interest has shifted on users547 home area. We then examined an archived UNIX Shell history file (.bash_history) for this particular user, kept in a non-user accessible location by the system administrator

```
1 ls
2 cd /var
3 touch test
```

In these three commands, the user attempts to see if there write permissions on the /var directory, by trying to create a test file. This particular operation would fail, since at the top level of /var/tmp only the super-user (root) accounts write access.

```
4 man dd
5 ls
6 ls -la
7 cd lock
8 ls
9 touch test
10 ls -la
11 cd ..
12 ls -la
13 cd tmp
14 ls
15 pwd
16 touch test
17 ls -la test
```

18 rm test

These commands help us start shaping a better picture of the scene. Here, user547 tries to identify useful options for a system level administrative command that performs low-level file manipulation (4). It then moves down to several subdirectories (always under the */var/tmp* subdirectory), trying to find a suitable place where he has write access (5-15). Eventually, he discovers that */var/tmp* can provide him with write access and he creates a file to test for write access permissions. When he is convinced about the ability to write he removes the 'test' file (16-18).

```
19 ls -la /dev/zero
20 dd if=/dev/zero of=/var/tmp/jj123-ssh
21 cat /var/log/messages
```

These commands are very indicative of the user intentions. Command no.20 creates (when non-interrupted) a file composed of null (*\0*) characters that can fill the entire partition. It is a quick and easy way to fill up disk space (or test disk performance in other circumstances). Command no.21 probably justifies the user to see what was logged from the system. If the partition becomes full, *syslogd* would report this fact to the 'messages' log file. However, if the */var* partition becomes full, *syslogd* will not be able to log any kind of information. Hence, the fact that the user examined this log file after performing the *dd* operation could potentially mean that he was trying to establish whether filling up the */var* partition managed to stop the logging daemon from doing its job or not.

```
22 rm /var/tmp/jj123-ssh
23 sync
24 df -h
25 cd ~
26 vi .testflags
27 chmod 700 .testflags
28 nohup ./testflags
29 exit
```

After removing the created file, the user then moves back to his home area and creates a script. He then decides to execute this script with *nohup* option (the job continues to execute after the user exits his shell session). The content of the script is very indicative of the user's intentions and is analyzed below.

```
1 touch $HOME/resultat
2 dd if=/dev/zero of=/var/tmp/jj123-ssh > /dev/null
3 dd if=/dev/zero of=$HOME/.fillupwithnots > /dev/null
4 $HOME/cloaxck $HOME/nohup.out $HOME/bash_history $HOME/.testflags
```

It is not clear what was the intention with line 1. Lines 2 and 3 show the application of the *dd* utility to fill up first the partition of the root partition (where */var* resides, so that *syslogd* will be disabled) and then the entire */home* area (where all the user directories reside).

When the desired actions have been achieved, line 4 employs a known utility that will securely erase a number of files, in order to destroy any evidence of his actions. The terms 'securely erase' mean that the files will be truly erased from the hard disk, so that 'file undelete' utilities will not be able to recover them. Should the malicious insider had invoked the shell's 'rm' command, the files would not have been accessible deleted, but it might have been possible to recover them with a suitable file recovery utility. Hence, the user tried to erase the *nohup.out* file (created by the fact he started the execution of his malicious shell script and then he logged off), his shell history file (that would reveal all his suspicious investigations) and finally the malicious script itself.

There were additional steps to ensure that these files were indeed created by this particular user and not by someone masquerading as user547. This involved the process of cross-referencing data from the LINUX 'lastlog' utility (which associates a particular user login to a date/time and workstation IP address) and information that verified that the particular person was physically present and using a workstation at that particular time.

This would have been the perfect insider attack. However, the attacker made two serious mistakes that prevented him from erasing the traces of his actions. Firstly, he mistyped the name of his shell history file (.bash_history as opposed to bash_history that he typed). This caused the cloaxck utility to abort, so neither the .bash_history nor the .testflags malicious script was erased. The second factor that connected the incident with this particular user is the fact that the large files that filled the system's /root and /var partitions carried the identity of user547. However, should the malicious insider have been successful in erasing the script and the shell history file, it would have been impossible to discover if he was really responsible for the incident or not.

Conclusions:

The incident shows how the operation of a relatively healthy system can be discontinued from an insider. Although many system administrators could argue that a departmental file server should always have a 'disk quota' mechanism installed and that /var and /root partitions should be separate, the incident has clearly indicated that:

- A knowledgeable insider can find ways to disrupt the operation of a system, exploiting its weak points.
- The applications that a particular insider executes and the impact they have on the computational resources of a system (storage space, memory, CPU time) could potentially indicate their level of knowledge and their intentions.

Appendix E: ITPM system source code

E1) Global monitoring script

```
#!/usr/bin/perl -w

#This is the global monitoring script -- Globalmon -- Version 1.2
beta
#(C) By George B. Magklaras - November 2002

#Essential Sanity checks...
@whoami=getpwuid($<);
die "Error:You should execute this program ONLY with root privileges.
You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);

#START/END SHELL SESSION MARKERS
#Strings used to mark the beginning and the end of
#a user's interactive shell session. They are vital
#in quantifying metrics such as FBreadth etc...
#--PS: This technique is still NOT EFFECTIVE for MULTIPLE SESSIONS
#      I NEED TO IMPLEMENT A GLOBAL MONITOR via the ps command.
# AND NOW I HAVE -- Dec 2003
$STARTMARK="###BEGSHELLSESSION###";
$ENDMARK="###FINSHELLSESSION###";

#Check if there is another monitoring process running.
#There should be ONLY one.

#$running=`ps auxwww | grep globalmon.pl | grep -v grep | grep -v vi
| wc -l`;
#debug
#print "running has a value of $running .\n";

#die "Error: Exiting, there is another globalmon process already
running.\n"
#"if ($running ne "1");

#If there is not another process running, did the previous one
#exit in a clean way?

die "Error: Exiting, the previous globalmon process did NOT exit
properly. Rerun itptmon stop.\n"
if (-e "/var/run/itptmon.pid");

#At this point, we should be OK, so we can start initialising the
environment
die "Error: Exiting, cannot access the pid control file.\n"
if (!(open PIDCONTROL, ">/var/run/itptmon.pid"));

$loopcontrolvar=1;

while ($loopcontrolvar) {

@loggedinusers=`ps auxwww | cut -d' ' -f1 | grep -v USER| grep -v
root | uniq `;

foreach $user (@loggedinusers) {
    chomp $user;
```

```

    }

print "itptmon has started.\n";

foreach $loggeduser (@loggedinusers) {
    $username=$loggeduser;
    if (!( -e "/var/log/.analysis$username" )) {
        unless (open ANALYSIS, ">/var/log/.analysis$username") {
            die "Error: Cannot create the analysis file for
user $username due to: $!";
        }

        $startdatestring=`date +%Y%m%d%H%M%S`;
        $yearofstart=substr($startdatestring,0,3);
        $monthofstart=substr($startdatestring,4,2);
        $dayofstart=substr($startdatestring,6,2);
        $hourofstart=substr($startdatestring,8,2);
        $minofstart=substr($startdatestring,10,2);
        $secofstart=substr($startdatestring,12,2);

        print ANALYSIS "###$STARTMARK\n";
        print ANALYSIS "###AAAYEAR:$yearofstart\n";
        print ANALYSIS "###AAAMONTH:$monthofstart\n";
        print ANALYSIS "###AAADAY:$dayofstart\n";
        print ANALYSIS "###AAAHOUR:$hourofstart\n";
        print ANALYSIS "###AAAMIN:$minofstart\n";
        print ANALYSIS "###AAASEC:$secofstart\n";
        print ANALYSIS "###AAAZZZZZZZZZZZZZZ\n";
        close ANALYSIS;
        #Now that we have closed the file descriptor we can fork
        #the monitoring scripts.
        #First the cmdparser.pl
        #This bit of code here uses the UNIX process management model
        #It will almost certainly need modification for Windows 2000/XP
        defined(my $pid=fork) or die "Error: Cannot kickstart cmdparser
due to: $!";
        unless ($pid) {
            exec "/usr/local/bin/cmdparser.pl", "$username";
            die "Error: Cannot kickstart cmdparser due to: $!";
        }
        #Maybe I should put some DoS protection code here
        #just in case something attempts to fork too many processes!

    } else {
        #Well if the file exists, it is best to do nothing, as the
        handling of the ENDMARK
        #insertion is done further down on another loop. Here we set a
        bogus variable.
        $fexists=1;
    }
} #end of foreach loop for loggedusers data

#Now rest a bit and then check if the same users are still logged in
#It is important that we 'system' the sleep command
#in order to force the program to wait.
system "sleep 5";

```

```

@loggedinusersb=`ps auxwww | cut -d' ' -f1 | grep -v USER| grep -v
root | uniq `;

foreach $user (@loggedinusersb) {
    chomp $user;
}

#And now see who has logged out. We can use PERL's grep and map
#specialist loops to achieve this.
#However the fastest way is to build a hash of the loggedinusersb
array
#and use it as a lookup table.

%lookup=();
@notloggedanymore=(); #(belonging to the loggedinusers array only)

foreach $currentlylogged (@loggedinusersb) {
    $lookup{$currentlylogged} = 1;
}
foreach $checkeduser (@loggedinusers) {
    unless ($lookup{$checkeduser}) {
        push(@notloggedanymore, $checkeduser);
    }
}

#How many analysis files do we have (EXCLUDING the .analysisb and
.analysisc files!)?
#@listoffiles=chomp(`ls -l .analysis* | grep -v .analysisb* | grep -v
.analysisc*`);

foreach $loggedoffuser (@notloggedanymore) {
    unless (open ANALYSIS, ">>/var/log/.analysis$loggedoffuser") {
        die "Error: Could not mark the end of the session for
user $loggedoffuser due to: $!";
    }
    $enddatestring=`date +%Y%m%d%H%M%S`;
    $yearofend=substr($enddatestring,0,3);
    $monthofend=substr($enddatestring,4,2);
    $dayofend=substr($enddatestring,6,2);
    $hourofend=substr($enddatestring,8,2);
    $minofend=substr($enddatestring,10,2);
    $secofend=substr($enddatestring,12,2);

    print ANALYSIS "$ENDMARK\n";
    print ANALYSIS "###ENDYEAR:$yearofend\n";
    print ANALYSIS "###ENDMONTH:$monthofend\n";
    print ANALYSIS "###ENDDAY:$dayofend\n";
    print ANALYSIS "###ENDHOUR:$hourofend\n";
    print ANALYSIS "###ENDMIN:$minofend\n";
    print ANALYSIS "###ENDSEC:$secofend\n";
    print ANALYSIS "###ENDZZZZZZZZZZZZZZZZZZZZ\n";
    close ANALYSIS;
}

} # end of while loop

```


E2)Command Register Script

```
#!/usr/bin/perl -w

#cmdregister script -- Version 1.8beta, (C) George B. Magklaras --
JUNE 2003
#SYNOPSIS: This script does most of the work. It extracts the
commands from
#/var/log/secure (where snoopy does its logging by default), taking
care and correlating
#data from multiple sessions on a single host. It takes a single
argument, which is the user name
#and when the user logs out it produces a session that contains all
the user commands
#in shell history file form.

# CHANGELOG: From version 1.7beta to version 1.8beta
#- Small re-design of the implementation of the Fdepth function. We
know report to the
# $USRNAMEssid file only the following data:
# noofcommands (strictly per session data)
# Fbreadth (strictly per session data and then averaged)
# totappscore (fappscore is going to average commands
# from all sessions)
# SCPU (fappscore is going to average CPU util
# from all sessions)
# SRAM (the same as SCPU)
# SIMAPPS (strictly per session data and then averaged)
# Then the usermon function will have the task of averaging the
numbers and evaluating
# fappsscore and other data, bringing it to the database (a file for
now and later an RDBMS).
# CHANGELOG: From version 1.6beta to version 1.7beta
#- An additional fault has been discovered with the Fbreadth
function. After inspecting
# manually a number of $USRNAMEssid files, they all had the same
value for Fbreadth (3).
# This was due to incorrect conditional expressions. Fixed. Also
included some extra weights
# on the constants for Fbreadth, to accommodate for the start and
end shell session commands
# (things such as dircolors, startsession, endsession, clear, etc).
# CHANGELOG: From version 1.5beta to version 1.6beta
#- A fault has been discovered in the data capture of Fdepth. Due to
the fact that
# the 'ps' command reports PIDs of 4 or 5 digits long, the shell
'cut' commands used to
# populate the cpuutilav and ramutilav arrays may contain blank
fields because of
# 'ps' output misalignments. Fixed that by adjusting the field
switch -f for 'cut'. There
# is probably a better way, but this is a quick and dirty hack.
#- Also on the Fdepth calculations, the normalisation of SCPU, SRAM
and SSIMAPPS were
# totally missing. Fixed by adding the necessary constants.
# CHANGELOG: From version 1.4beta to version 1.5beta
#- In version 1.4 beta, the implementation of the various
Fdepth/Fbreadth metrics is really
# bad to mediocre in the best case scenario. Addressed that issue
with various changes.
```

```

# In particular, the appscores hash has been updated with a more
complete list of command
# scores, and FDepth has been moved a bit further down the
computational process for the
# purposes of efficiency.
#- On the final stage, where we locate the point where the old and
the freshly acquired
# user data coincide, I was shifting the arrays in the bogus array.
This is unnecessary
# and is a waste of RAM. I am now placing the result of the shift to
a bogus variable.
# CHANGELOG: From version 1.3beta to version 1.4beta
#- Fixed an alignment problem between @usrcommands and
@commandsarguments due to
# incorerct order of performing the regexps between the two. The
order should
# have been maintained.
#- An additional error has been discovered due to the wat we extract
information from
# /var/log/secure. The idea is that the $USRNAMEsessionid file
should contain info
# for one use session. However, it doesn't, since information from
previous sessions
# is maintained in /var/log/secure. Fixed that by doing some
additional checking.
#- Corrected various error messages to indicate certain abnormal
conditions so that they
# are more accurate or meaningful.
# CHANGELOG: From version 1.2beta to version 1.3beta
#- Fixed command line argument processing which was missing on
version 1.2beta
#- Cleaned various unnecessary stdout debug statements.

#Essential Sanity checks...
@whoami=getpwuid($<);
die "Error:You should execute this program ONLY with root privileges.
You are not root.\n"
if ($whoami[2] != 0 && $whoami[3] != 0);

die "Usage: cmdparser [username] . You MUST specify a valid
username.\n"
if (($ARGV[0] eq ""));

#Is a globalmon process running. If not, we are in trouble and we
should not be running

#$globalmonrun=`ps auxwww | grep globalmon.pl | grep -v grep | wc -
l`;

#die "cmdparser for $ARGV[0] : Error: No single globalmon process
running. Exiting. \n"
#if ( $globalmonrun ne "1");

#Scoring constants. Here we introduce important constants for
normalising the
#sophistication scoring of the users. This beta version of the script
has these values
#hard-wired by means of constants and hash assignments. The values
can be derived by running
#the script with the 'train' option. This will calculate average non
normalised values, for

```

```
#certain categories of users. I have hardwired these values now, but
in later versions, I
#plan to obtain them from RDBMS SQL queries.
```

```
#Fbreadth
```

```
#Normally it would be 7 for novices, 11 for masters and 14 for
advanced users.
#But we add to all these numbers a weight of +4 to accommodate for
things such as start
#and end shell session commands. In particular, every command
sequence will always have
#the following unique commands, executed automatically by start and
finish session shell
#scripts: /sbin/consoletype, /usr/bin/dircolors --sh /etc/DIR_COLORS,
#/usr/bin/startshellsession, /usr/bin/locale charmap, /bin/uname -m,
/usr/bin/dumpkeys,
# are always at the start, and /usr/bin/endshellsession,
/usr/bin/clear at the end.
# That makes a total of 8, hence the weight of +8. This is OS (or
even HOST) specific and
# this issue should be tackled by the cmdparser script in record
mode!!!
$MAXDIFFAPPSFORNOVICES=15;
$MINDIFFAPPSFORADVANCED=22;
$MAXDIFFAPPSFORMASTERS=19;
```

```
#Fdepth
```

```
#i)Fappsscore
```

```
#Best to use a hash to assign a score to the applications
```

```
%appscores=(
  "/usr/bin/startshellsession" => "0.5",
  "/usr/bin/endshellsession" => "0.5",
  "/usr/bin/clear" => "0.5",
  "/bin/cat" => "0.5",
  "cat" => "0.5",
  "/bin/ls" => "0.5",
  "ls" => "0.5",
  "/bin/vi" => "0.5",
  "/bin/cp" => "0.5",
  "/usr/bin/mozilla" => "0.5",
  "/usr/bin/xmms" => "0.5",
  "xmms" => "0.5",
  "/usr/bin/less" => "0.5",
  "/bin/date" => "0.5",
  "/usr/bin/ssh" => "0.5",
  "/bin/mkdir" => "0.5",
  "/bin/ping" => "1",
  "ping" => "1",
  "/usr/bin/gcc" => "3",
  "gcc" => "3",
  "/usr/bin/cc" => "3",
  "/usr/sbin/tcpdump" => "3",
  "tcpdump" => "3",
  "/bin/awk" => "3",
  "/usr/bin/perl" => "1",
  "/usr/bin/diff" => "1",
  "/usr/bin/who" => "0.5",
  "/bin/id" => "0.5",
  "/bin/hostname" => "0.5",
  "/bin/grep" => "1",
```

```

        "/bin/fgrep" => "1",
        "/bin/chgrp" => "1",
        "/bin/chmod" => "1",
        "/usr/bin/make" => "1",
        "/bin/tar" => "1",
    );

#ii)Fresutil
#Values represent %values as reported by 'ps'
$MAXCPUUTILFORNOVICES=0.7;
$MAXCPUUTILFORORDINARIES=5.5;
$MINIMUMCPUUTILFORADVANCED=15.0;
$MAXMEMUTILFORNOVICES=0.1;
$MAXMEMUTILFORORDINARIES="test";

$USERNAME=$ARGV[0];

#Fish the uid and gid. An essential step for many operations
#to follow:

$susruuid=getpwnam($USERNAME);
$susrgid=getgrnam($USERNAME);

die "Error: Username $USERNAME does not exist.\n"
if (!(defined($susruuid)));

#debug
#print "Fetched uid for user $USERNAME was $susruuid\n";

#START/END SHELL SESSION MARKERS
#Strings used to mark the beginning and the end of
#a user's interactive shell session. They are vital
#in quantifying metrics such as FBreadth etc...
#--PS: This technique is still NOT EFFECTIVE for MULTIPLE SESSIONS
#      I NEED TO IMPLEMENT A GLOBAL MONITOR via the ps command.
#AND NOW I HAVE -- Dec 2003

$STARTMARK="###BEGSHELLSESSION###";
$ENDMARK="###FINSHELLSESSION###";

system "cat /var/log/secure | grep '$USERNAME, uid:$susruuid' | grep -v
-i accepted | grep -v from | grep -v '(null)' >>
/var/log/.analysis$USERNAME";
#Does .analysis$USERNAME display an end session tag?
#debug
print "cmdparser on $USERNAME: About to enter loop.\n";
#Note the use of eval here. This is of particular importance
#when we wish to assign a value in the EXPR of the conditional
statement.
#If we just used the backticks to assign the result to a scalar
variable
#thinking that $scalar would be returned, IT WONT!! IT RETURNS
SUCCESS
#OR FAILURE IN ASSIGNING THE BACKTICK VALUES TO THE VARIABLE AND
WON'T PRODUCE THE
#RIGHT RESULT. ALWAYS USE AN EVAL IN IF/WHILE...conditions!

```

```

while ( (eval `grep "$ENDMARK" /var/log/.analysis$USRNAME | wc -l`)
ne "1") {
    #Take a snapshot of computational resource utilisation data:
    @cpuutilav=`ps u | grep $USRNAME | grep -v ^USER | cut -d' ' -
f8-9`;
    @ramutilav=`ps u | grep $USRNAME | grep -v ^USER | cut -d' ' -
f10-11`;
    #Taking the simultaneous number of apps is not necessary, as it
can be
    #sensed by the array size of either cpuutilav or ramutilav
outside this
    #computational loop for efficiency.

    #Has anything changed, any new commands?
    system "sleep 2";
    #If .analysisb$USRNAME exists we have hit a race condition and
this is an exception
    die "Exiting, due to race condition with user $USRNAME \n"
    if ( -e "/var/log/.analysisb$USRNAME");
    system "cat /var/log/secure | grep '$USRNAME, uid:$susruid' |
grep -v -i accepted | grep -v from | grep -v '(null)' >
/var/log/.analysisb$USRNAME";
    #debug
    print "cmdparser on $USRNAME: Executing /var/log/secure second
round of greps for analysisb.\n";
    system "sleep 2";
    #The same for .analysisc$USRNAME
    die "Exiting, due to race condition with user $USRNAME \n"
    if ( -e "/var/log/.analysisc$USRNAME");
    #debug
    print "cmdparser on $USRNAME: Executing third round of greps
for analysisc.\n";
    system "cat /var/log/secure | grep '$USRNAME, uid:$susruid' |
grep -v -i accepted | grep -v from | grep -v '(null)' >
/var/log/.analysisc$USRNAME";
    system "cat /var/log/.analysis$USRNAME
/var/log/.analysisb$USRNAME /var/log/.analysisc$USRNAME | sort | uniq
> /var/log/.analysis$USRNAME";
    system "rm -f /var/log/.analysisb$USRNAME
/var/log/.analysisc$USRNAME";
}

```

```

#At this point, the user must have exited the session and we should
be ready to collect
#the session data.

```

```

@dateofcommands=`cat /var/log/.analysis$USRNAME | grep -v -i
identification | grep -v ^### | cut -b1-15`;
@usrcommands=`cat /var/log/.analysis$USRNAME | grep -v -i
identification | grep -v ^### | cut -b 16- | cut -d' ' -f7 | grep -v
"$STARTMARK" | grep -v "$ENDMARK"`;

```

```

@commandsarguments=`cat /var/log/.analysis$USRNAME | grep -v -i
identification | grep -v ^### | cut -b 16- | cut -d' ' -f8- | grep -v
"$STARTMARK" | grep -v "$ENDMARK"`;

```

```

foreach $dataarray (@dateofcommands, @usrcommands,
@commandsarguments) {
    foreach $arrayelement ($dataarray) {
        chomp $arrayelement;
    }
}

```

```

    }
}

$sizeofcommandset=$#usrcommands;
$noofcommands=$sizeofcommandset+1;

@sessfiles=glob "/var/log/$USRNAME*";

if ( $#sessfiles != -1 ) {
    print "Entering loop where old session file was detected. \n";
    @oldsessiondateofcommands=`cat /var/log/$USRNAME* | grep -v
^### | grep -v ^Recorded | cut -d' ' -f1-3`;
    @oldsessionusrcommands=`cat /var/log/$USRNAME* | grep -v ^### |
grep -v ^Recorded | cut -d' ' -f4`;
    @oldsessioncommandsarguments=`cat /var/log/$USRNAME* | grep -v
^### | grep -v ^Recorded | cut -d' ' -f5-`;

    foreach $dataarray (@oldsessiondateofcommands,
@oldsessionusrcommands, @oldsessioncommandsarguments) {
        foreach $arrayelement ($dataarray) {
            chomp $arrayelement;
        }
    }

    #Find at which point the new commands arrays coincide (aka at
which point the last dateofcommand
    #from the old session is located at the dateofcommands from
the freshly acquired data.)

    $sizeofoldsession=$#oldsessiondateofcommands;
    $stringtolocate="$oldsessiondateofcommands[$sizeofoldsession]
$oldsessionusrcommands[$sizeofoldsession]
$oldsessioncommandsarguments[$sizeofoldsession]";

    #And now start comparing from the end of the freshly acquired
data, in order
    #to increase runtime efficiency (the most recent data are
closer to the end of the arrays).
    #Be careful! index is a reserved keyword in PERL. Also take
care in the conditional to have
    #at least expression of the type >= <= in specifying the
limits. Otherwise the loop will
    #hang!
    for ($loopindex=$sizeofcommandset; $loopindex>=0; $loopindex--)
    {
        $linestring="$dateofcommands[$loopindex]
$usrcommands[$loopindex] $commandsarguments[$loopindex]";
        if ($linestring eq $stringtolocate) {
            $indexlocation=$loopindex; } else {
$bogus=3;}
    }

    #So everything from 0 to indexlocation is old and should vacate
the premises
    #on the dateofcommands usrcommands and commandsarguments
arrays.

    for ($n=0; $n<=$indexlocation; $n++) {
        $bogus=shift @dateofcommands;
        $bogus=shift @usrcommands;

```

```

        $bogus=shift @commandsarguments;
    }

    #Analysis of Fbreadth
    %seen = ();
    foreach $item (@usrcommands) {
        $seen{$item}++;
    }
    @fbreadtharray = keys %seen;

    $noofdifffapps=$#fbreadtharray;
    #debug
    print "No. of different applications: $noofdifffapps \n";

    if ($noofdifffapps <= $MAXDIFFAPPSFORNOVICES) { $fbreadth=1;}
elseif
    ($noofdifffapps > $MAXDIFFAPPSFORNOVICES && $noofdifffapps <
    $MINDIFFAPPSFORADVANCED) { $fbreadth=3;} else {$fbreadth=6;}

    #Analysis of FDepth
    #i)Fappstypescore
    $totappscore=0;
    foreach $commands (@usrcommands) {
        $totappscore+=$appscores{$commands};
    }
    $fappstypescore=$totappscore/$noofcommands;
    #ii)Fresourceutil
    #debug
    print "Cpuutilav matrix has the following elements: @cpuutilav
\n";
    print "Ramutilav matrix has the following elements: @ramutilav
\n";

    for ($loopindex=0; $loopindex<=$#cpuutilav; $loopindex++) {
        $SCPU+=$cpuutilav[$loopindex]/$#cpuutilav;
        $SRAM+=$ramutilav[$loopindex]/$#ramutilav;
    }

    #This check is necessary in case we have parsing problems with
the
    #resource utilisation data. Will be removed if the parsing
proves to
    #be stable.
    if ($#cpuutilav == $#ramutilav) { $SIMAPPS=$#cpuutilav;} else
{ $SIMAPPS=0;}

    $fdepth=$fappstypescore+$SCPU+$SRAM+$SIMAPPS;

    $sessfileid=`date +%Y%m%d%H%M%S`;
    #to avoid a gap line, chomp the newline.
    chomp $sessfileid;

    unless (open SESSFILE, "> /var/log/$USERNAME$sessfileid") {
        die "cmdparser Error: Could not create the session file
for user $USERNAME due to: $!";
    }

    print SESSFILE "Recorded: $sessfileid \n";
    $datedata=`grep '###' /var/log/.analysis$USERNAME`;
    print SESSFILE $datedata;

```

```

print SESSFILE "###---NOOFCOMMANDS=$noofcommands\n";
print SESSFILE "###---FBREADTH=$fbreadth\n";
print SESSFILE "###---TOTAPPSCORE=$totappscore\n";
print SESSFILE "###---SCPU=$SCPU\n";
print SESSFILE "###---SRAM=$SRAM\n";
print SESSFILE "###---SIMAPPS=$SIMAPPS\n";

for ($loopindex=0; $loopindex <= $#usrcommands; $loopindex++) {
    print SESSFILE "$dateofcommands[$loopindex]
$usrcommands[$loopindex] $commandsarguments[$loopindex]\n";
}

close SESSFILE;
system "rm -f /var/log/.analysis$USRNAME";
#debug
print "Cmdparser exiting for user $USRNAME \n";

} else {
    $sessfileid=`date +%Y%m%d%H%M%S`;
    #to avoid a gap line, chomp the newline.
    chomp $sessfileid;

    #Analysis of Fbreadth
    %seen = ();
    foreach $item (@usrcommands) {
        $seen{$item}++;
    }
    @fbreadtharray = keys %seen;

    $noofdiffapps=$#fbreadtharray;
    if ($noofdiffapps <= $MAXDIFFAPPSFORNOVICES) {
$fbreadth=1;} elseif
    ($noofdiffapps > $MAXDIFFAPPSFORNOVICES && $noofdiffapps
< $MINDIFFAPPSFORADVANCED) { $fbreadth=3;} else {$fbreadth=6;}

    #Analysis of FDepth
    #i)Fappstypescore
    $totappscore=0;
    foreach $commands (@usrcommands) {
        $totappscore+=$appscores{$commands};
    }
    $fappstypescore=$totappscore/$noofcommands;
    #ii)Fresourceutil
    #debug
    print "Cpuutilav matrix has the following elements:
@cpuutilav \n";
    print "Ramutilav matrix has the following elements:
@ramutilav \n";

    for ($loopindex=0; $loopindex<=$#cpuutilav; $loopindex++)
    {
        $SCPU+=$cpuutilav[$loopindex]/$#cpuutilav;
        $SRAM+=$ramutilav[$loopindex]/$#ramutilav;
    }
    #This check is necessary in case we have parsing problems
with the
    #resource utilisation data. Will be removed if the
parsing proves to
    #be stable.

```



```

        if ($#cpuutilav == $#ramutilav) {
$SIMAPPS=$#cpuutilav;} else { $SIMAPPS=0;}
        $fdepth=$fappstypescore+$SCPU+$SRAM+$SIMAPPS;

        unless (open SESSFILE, "> /var/log/$USRNAME$sessfileid")
{
        die "cmdparser Error: Could not create the session
file for user $USRNAME due to: $!";
        }

        print SESSFILE "Recorded: $sessfileid \n";
        $datedata=`grep '###' /var/log/.analysis$USRNAME`;
        print SESSFILE $datedata;

        print SESSFILE "###---FBREADTH=$fbreadth\n";
        print SESSFILE "###---FDEPTH=$fdepth\n";

        for ($index=0; $index <= $#usrcommands; $index++) {
                print SESSFILE "$dateofcommands[$index]
$usrcommands[$index] $commandsarguments[$index]\n";
        }

        close SESSFILE;

        system "rm -f /var/log/.analysis$USRNAME";
        #debug
        print "Cmdparser exiting for user $USRNAME \n";
}

```

E3)Realtime monitoring script

```

#!/usr/bin/perl

# realltimemon - (C) 2003 George B. Magklaras
# version 1.72beta- Removed all debug statements from 1.71 and
inserted memory debug
#         measurement statements.
# Version 1.71beta- Included a check on the inner loop, in order to
test for the
#         size of the asignature array (if it is 0, then we
should not
#         enter at all the inner loop.
#         Also included the match score print statements in two
places.
#         Once, where the algorithm terminates in the worst
case scenario
#         with no matches, and once where the algorithm ends
where matches
#         are found. (two possible exit points for the
algorithm).
# Version 1.7beta - Removed most of the debug statements in order to
analyse
#         the performance of the algorithm.
# Version 1.6beta - Enforced a 'use strict' interface totidy up a bit
the mess
#         Finish off the sequence comparison algorithm.

```

```

# Version 1.5beta - Modified the command termination sequence
# see comments on the createsequence1point5beta version tool.

#use Env;
use strict;

#START/END SHELL SESSION MARKERS
#Strings used to mark the beginning and the end of
#a user's interactive shell session.
my $STARTMARK="###BEGSHELLSESSION###";
my $ENDMARK="###FINSHELLSESSION###";

#The value the encoder will use for a command that has not
#being registered.
my $UNKNOWNCMD="-1";

chomp(my $localhostname=`/bin/hostname`);
chomp(my $localtargetos=`/bin/uname -rsm`);
my $localsignaturedir="/var/log/signatures$localhostname";

die "createsequence Error: The directory $localsignaturedir does not
exist. Register the host first.\n"
if (!( -e "$localsignaturedir" ));

#Essential Sanity checks...
my @whoami=getpwuid($<);
die "realtimemon Error:You should execute this program ONLY with root
privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);

die "Usage: realtimemon [username] . You MUST specify a valid
username.\n"
if (($ARGV[0] eq ""));

my $ONLINEUSER=$ARGV[0];
my $onlineusruid=getpwnam($ONLINEUSER);
my $onlineusrgid=getgrnam($ONLINEUSER);

die "Error: Username $ONLINEUSER does not exist.\n"
if (!(defined($onlineusruid)));

my $hostname=$localhostname;

#Has this host ran successfully the commandregister script.
die "realtimemon Error: Could not find the command enumeration codes.
Have you run the commandregister script?\n"
if (!( -e "/var/log/regcommandson$hostname" ));

#Read the command codes from the file into a hash.
unless (open REGCOMMHR, "</var/log/regcommandson$hostname") {
    die "realtimemon Error: Cannot read the registered
commands file for host $hostname due to: $!";
}

# Note that because the registered commands file has white space as a
field
# delimiter, we will need to 'split' the stream from the REGCOMMHR
# file hande with the same single whitespace character.
my %readcommhash=split(" ", <REGCOMMHR>);

close REGCOMMHR;

```

```

#At this point, we have the command codes, and we are ready to start
the processing.

#Do we have a .analysis$ONLINEUSER file?
#If yes, encode the user commands. If not,
#exit with an error message.

if ( -e "/var/log/.analysis$ONLINEUSER" ) {
    my @onlinecmds=`cat /var/log/.analysis$ONLINEUSER | grep -v -i
identification | grep -v ^### | cut -b 16- | cut -d' ' -f7 | grep -v
"$STARTMARK" | grep -v "$ENDMARK"`;
    my @onlinecmdargs=`cat /var/log/.analysis$ONLINEUSER | grep -v
-i identification | grep -v ^### | cut -b 16- | cut -d' ' -f8- | grep
-v "$STARTMARK" | grep -v "$ENDMARK"`;
    #Encode the commands
    my $onlinecmd;
    my $cmdtopush;
    my @encodedcmds;
    foreach $onlinecmd (@onlinecmds) {
        chomp $onlinecmd;
        #Does it exist in the hash?
        if (exists $readcommhash{$onlinecmd}) {
            $cmdtopush=$readcommhash{$onlinecmd};
            push(@encodedcmds, $cmdtopush);} else {
                push(@encodedcmds, $UNKNOWNCMD);
            }
        }
    my $onlinecmdarg;
    my @encodedcmdargs;
    foreach $onlinecmdarg (@onlinecmdargs) {
        if (($onlinecmdarg eq " ") ||
        (!(defined($onlinecmdarg)))) {
            push(@encodedcmdargs, "noarg"); } else {
                chomp $onlinecmdarg;
                push(@encodedcmdargs, $onlinecmdarg);
            }
        }
    #Do we have as many arguments as encoded commands?
    die "realtimeon Fatal Error: Sorry, the encoding process has
failed for user $ONLINEUSER \n"
    if(! ($#encodedcmds==$#encodedcmdargs));

    #Then just output the sequence in stdout.
    my $loopcount;
    my $stringtopush;
    my @sequence;
    for ($loopcount=0; $loopcount <= $#encodedcmds; $loopcount++) {
        #Note that here, we do not need to employ the sequence
        termination string
        #-##8# , because this is only necessary when we parse the
        stored misuse
        #signatures. Here, we acquire the data straight from the
        snoopy log file
        #and hence we use only the
        CcommandcodeAargument1argument2.. format

        $stringtopush="C$encodedcmds[$loopcount]A$encodedcmdargs[$loopc
ount]";
        push(@sequence, $stringtopush);
    }
}

```

```

}

my $maxsequenceindex=$#sequence - 1;
#Debug
#print "The sequence is array is: @sequence ";
my $seqsize=$#sequence;
#print "with size $seqsize \n";

#Now, read the misuse signatures in the local signature
directory:
    opendir MISDIRH, $localsignaturedir or die "realtimemon Error:
Could not open the misuse signature dir: $! \n";
    my $nameoffile;
    my @listofsignfiles;
    while ($nameoffile=readdir MISDIRH) {
        #Skip over the . and .. files (standard in each dir)
        next if $nameoffile=~/^\.\/;
        #insert the absolute path of the misuse signature files
so that
        #the next loop will access the files directly.
        $nameoffile="$localsignaturedir/$nameoffile";
        push(@listofsignfiles, $nameoffile);
    }
    close MISDIRH;

#And now, lets start the real work. Take the extracted sequence
and
#compare it against all the misuse signatures for the host one
by one
    my $noofmatches=0;
    my $misusesignature;
    my $misusesesequence;
    my $loopcountj;
    my $noofsigcmds;
    my @signature;
    my $matchscore;
    my $totcomparisons;
    my $innerloopcmps;
    my $outerloopcmps;
    my $maxasignatureindex;

#Debug for memcons
my $memcons;

#Debug for memcons
$memcons=`ps auxwww | grep realtimemon.pl | grep -v grep`;
print "Before the outerloop: $memcons \n";

foreach $misusesignature (@listofsignfiles) {
    chomp($misusesesequence=`cat $misusesignature | grep -v
^###`);
    #convert the misuse signature string to an array.
    @signature=split /-##8#/, $misusesesequence;

    $noofsigcmds=$#signature;
    $maxasignatureindex=$noofsigcmds - 1;
    #Debug

    #print "Comparing against the $misusesignature signature
that has $noofsigcmds commands...\n";

```

```

        for ($loopcount=0; $loopcount <= $#sequence;
$loopcount++) {
            $outerloopcmps++;
            #debug:
            #print "Current loopcount for theuser sequence loop
is $loopcount . \n";
            if ($#asignature!=0) {
                for ($loopcountj=0; $loopcountj <= eval
{ $#asignature-1}; $loopcountj++) {
                    #Debug memcons
                    $memcons=`ps auxwww | grep realltimemon.pl | grep -v
grep`;
                    print "In the innerloop: $memcons \n";
                    #Debug
                    #print "$sequence[$loopcount] versus
$asignature[$loopcountj] \n";
                    $innerloopcmps++;
                    if ($sequence[$loopcount] eq
$asignature[$loopcountj]) {
                        $noofmatches++;
                                                                    shift
                    }
                }
            }
        }
    } else {
        #Debug
        #Here we print the total statistics in the case of the
algorithm
        #terminating NOT in the worst case scenario
        $matchscore=(100*($noofmatches/$noofsigcmds));
        print "Match score for $misusesignature is
(($noofmatches/$noofsigcmds)*100)=$matchscore \n";
        $totcomparisons=$innerloopcmps*$outerloopcmps;
        print "Innerloopcmps=$innerloopcmps ,
Outerloopcmps=$outerloopcmps, totalcomparisons=$totcomparisons \n";
    }
}
    #Here, we print the total statistics in the case of the
algorithm
    #terminating without finding a match (worst case
scenario).
    $matchscore=(100*($noofmatches/$noofsigcmds));
    print "Match score for $misusesignature is
(($noofmatches/$noofsigcmds)*100)=$matchscore \n";
    $totcomparisons=$innerloopcmps*$outerloopcmps;
    print "Innerloopcmps=$innerloopcmps ,
Outerloopcmps=$outerloopcmps, totalcomparisons=$totcomparisons \n";
}
} else {
    print "realltimemon Fatal Error: Sorry, the /var/log/.analysis
file for user $ONLINEUSER cannot be found.\n";
    return -1;
}
}

```

E4) 'createsequence' script

```
#!/usr/bin/perl -w

# createsequence version 1.5beta (C) 2004 George B. Magklaras
# A correction to the sequence encoding format was made in this
version
# in order to provide a better command separation sequence.
# Instead of the character S (CcommandcodeAcommandargsS) we now
terminate
# every encoded command with the string -##8# . So the encoding
scheme now becomes
# CcommandcodeAcommandargs-##8# . The realltimeon script required
the same change.
# createsequence version 1.4beta (C) 2004 George B. Magklaras
# Modified the program so that whitespace is removed before
# the arguments are encoded. This is necessary to improve the
# alignment algorithm, as the spaces might disorient the alignment
# algorithm.

use strict;
use DBI;

#The value the encoder will use for a command that has not
#being registered.
my $UNKNOWNCMD="-1";

#The string returned by whitch, if the command fails to find
#the path of a command.
my $NOWHICHPATH="watch: no";

# Essential Sanity checks...
my @whoami=getpwuid($<);
die "creatsequece Error:You should execute this program ONLY with
root privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);

#Do some essential data gathering on the local host.
chomp(my $localhostname=`/bin/hostname`);
chomp(my $localtargetos=`/bin/uname -rsm`);
my $localsignaturedir="/var/log/signatures$localhostname";

die "createsequence Error: The directory $localsignaturedir does not
exist. Register the host first.\n"
if (!(-e $localsignaturedir));

#Has this host ran successfully the commandregister script.
die "realltimeon Error: Could not find the command enumeration codes.
Have you run the commandregister script?\n"
if (!(-e "/var/log/regcommandson$localhostname"));

print "#####\n";
print "# createsequence itpmdb tool Version 1.5 beta #\n";
print "#####\n";
print "# (C) 2003 George B. Magklaras #\n";
print "#####\n";
print "#Enter the sequence consequence type: (max 20 chars) \n";
my $consequence=<STDIN>;
chomp $consequence;
die "createsequence Error: Sorry, that string is too long. Try again
with a shorter one. \n"
```

```

if (length($consequence)> 20);

print "#Enter the first misuse keyword (max 20 chars):\n";
my $firstmisuseword=<STDIN>;
chomp $firstmisuseword;
die "createsequence Error: Sorry, that string is too long. Try again
with a shorter one. \n"
if (length($firstmisuseword)> 20);
print "#Enter a second misuse keyword (max 20 chars):\n";
my $secmisusekeyword=<STDIN>;
chomp $secmisusekeyword;
die "createsequence Error: Sorry, that string is too long. Try again
with a shorter one. \n"
if (length($secmisusekeyword)> 20);
print "#Enter a third misuse keyword (max 20 chars):\n";
my $thirdmisusekeyword=<STDIN>;
chomp $thirdmisusekeyword;
die "createsequence Error: Sorry, that string is too long. Try again
with a shorter one. \n"
if (length($thirdmisusekeyword)> 20);

print "#Now enter the commands at the cmd prompt \n";
print "#and the respective arguments at the arg prompt.\n";
print "#Type the string END at the cmd prompt, to exit the \n";
print "#command input loop. \n";

my $loopcount=1;
my @inputcmds;
my @inputargs;
my $stcmd;
my $starg;

while ($stcmd ne "END") {
    print "%Enter cmd$loopcount:\n";
    $stcmd=<STDIN>;
    chomp $stcmd;
    # Now we need to find the full path of the command.
    # This is because "Snoopy" reports the full path in the logs.
    my $tempstcmd=`which $stcmd 2> /dev/null`;
    # Depending on whether which will locate the full path, the
program acts
    # accordingly, based on what 'which' normally returns if it
fails to locate.
    # the command. Here, I redirect STDERR to /dev/null so, if
which fails it
    # will return an empty string.

    if (!$tempstcmd eq "") {
        $stcmd=$tempstcmd;
        chomp $stcmd;
    }

    print "%Enter args$loopcount:\n";
    $starg=<STDIN>;
    chomp $starg;
    # It is necessary to collapse any white space in the arguments.
    $starg =~ s/\s+//g;
    push(@inputcmds, $stcmd);
    push(@inputargs, $starg);
    $loopcount=$loopcount+1;
}

```

```

#Do we have as many arguments as encoded commands?
die "createsequence Fatal Error: Sorry, the misuse signature creation
has failed during command input stage.\n"
if(! ($#inputcmds==$#inputargs));

die "createsequence Error: Sorry, but you entered no commands for the
misuse signature. Try again.\n"
if($#inputcmds==0);

print "#OK, got $#inputcmds commands with their respective
arguments.\n";

#OK, and we are now ready to encode the sequence.
print "#Encoding the sequence ...\n";

#Read the command codes from the file into a hash.
unless (open REGCOMMHR1, "</var/log/regcommandson$localhostname") {
    die "realtimemon Error: Cannot read the registered commands
file for host $localhostname due to: $!";
}

my %readcommhash=split(" ", <REGCOMMHR1>);

close REGCOMMHR1;

my @encodedcmds;
my @encodedargs;
my $onlinecmd;
my $onlinearg;
my $cmdtopush;

foreach $onlinecmd (@inputcmds) {
    if (exists $readcommhash{$onlinecmd}) {
        $cmdtopush=$readcommhash{$onlinecmd};
        push(@encodedcmds, $cmdtopush);} else {
        push(@encodedcmds, $UNKNOWNCMD);
    }
}

foreach $onlinearg (@inputargs) {
    if (($onlinearg eq " ") || (!(defined($onlinearg)))) {
        push(@encodedargs, "noarg"); } else {
        push(@encodedargs, $onlinearg);
    }
}

#Do we have as many encoded arguments as encoded commands?
die "createsequence Fatal Error: Sorry, the misuse signature creation
has failed during the encoding stage \n"
if(! ($#encodedcmds==$#encodedargs));

#Construct the misuse signature string
my $stringtopush;
my @finalsequence;
for ($loopcount=0; $loopcount <= $#encodedcmds; $loopcount++) {
    $stringtopush="C$encodedcmds[$loopcount]A$encodedargs[$loopcount]
-##8#";
    push(@finalsequence, $stringtopush);
}

```



```

}

#It is now time to connect to the database and get an ID that will
form the name
#of the file.

die "registerhost Error:No MySQL client cnf file found.\n"
if (! (-e "/root/.my.cnf"));

my
$datasource="DBI:mysql:itpmdb:mysql_read_default_file=/root/.my.cnf";

my $itpmservh=DBI->connect ($datasource, undef, undef, {RaiseError =>
1, PrintError => 1});

chomp(my $creationyear=`date +%Y`);
chomp(my $creationmonth=`date +%m`);
chomp(my $creationday=`date +%d`);
#The nanosecond time indicator is created here not for insertion on
to the ITPMdb.
#Its purpose is to provide a way of creating a unique filename
locally (a second
#resolution might not be enough.

chomp(my $creationnanosec=`date +%N`);

my
$localsignfile="sig$creationyear$creationmonth$creationday$creationna
nosec";
#Build a string and pass it to the OPEN statement because the dots
and / symbols
# might not be interpolated properly and cause problems.
my $signfiletoopen="$localsignaturedir/$localsignfile";

#Now it is wise to try and create the file locally first and then
update the database.
#In that way the database will not contain incorrect information.
unless (open SIGFILEH, ">$signfiletoopen") {
    die "createsequence Error: Could not open the local signature
file in $localsignaturedir due to: $! \n";
}

print SIGFILEH "###:$creationyear \n";
print SIGFILEH "###:$creationmonth \n";
print SIGFILEH "###:$creationday \n";
print SIGFILEH "###:$localtargetos \n";
print SIGFILEH "###:$localtargetos \n";
print SIGFILEH "###:$consequence \n";
print SIGFILEH "###:$firstmisuseword \n";
print SIGFILEH "###:$secmisusekeyword \n";
print SIGFILEH "###:$thirdmisusekeyword \n";
print SIGFILEH @finalsequence;

close SIGFILEH;

my $rows=$itpmservh->do("INSERT INTO signatures
(targetos,hostfqdn,consequencetype,misusekeyword1,misusekeyword2,misu
sekeyword3,creationday,creationmonth,creationyear,sigdir,signfile)"
. "VALUES
('$localtargetos','$localhostname','$consequence','$firstmisuseword',

```

```
'$secmisusekeyword','$thirdmisusekeyword','$creationday','$creationmonth',
'creationyear','$localsignaturedir','$localsignfile')");

print "#####\n";
print "createsequence STATUS: \n";
print "Created sequence on $localhostname with file name
$localsignfile of consequence: $consequence \n";
print "bound to the keywords: $firstmisuseword, $secmisusekeyword,
$thirdmisusekeyword .\n";
```

E5) Snoopy execve logger source code (C programming language)

```
/* snoopy.c -- execve() logging wrapper
 * Copyright (c) 2000 marius@linux.com,mbm@linux.com
 * Version 1.1
 * $Id: snoopy.c,v 1.5 2000/09/27 05:16:40 mbm Exp $
 *
 * Part hacked on flight KL 0617, 30,000 ft or so over the Atlantic
 :)
 *
 * This program is free software; you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License as published
by
 * the Free Software Foundation; either version 2, or (at your
option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */
/* Minor modifications by George B. Magklaras 02/2002 */
/* Mainly to modify the destination of the output file */

#include <dlfcn.h>
#include <stdio.h>
#include <syslog.h>
#include "snoopy.h"

#if defined(RTLD_NEXT)
# define REAL_LIBC RTLD_NEXT
#else
# define REAL_LIBC ((void *) -1L)
#endif

#define FN(ptr,type,name,args) ptr = (type (*)args)dlsym (REAL_LIBC,
name)

inline void log(const char *filename, char **argv) {

    static char **ptr, *logstring;
    static int size = MAX;
```

```

static int (*guid)(void);

FN(guid,int,"getuid", (void));

ptr      = (char **) &argv[1];
logstring = (char *) malloc((size_t *) size+2);

/* Here, we change the default openlog function call
   from what is shown below, to redirect the cause
   syslog to redirect the output to a specific file.
   The logging codes are in </sys/syslog.h>.

   openlog("snoopy", LOG_PID, LOG_AUTHPRIV); */
openlog("ITPMhostlogger", LOG_PID, LOG_LOCAL3);

size -= sprintf(logstring, size, "[%s, uid:%d sid:%d]: %s",
               getlogin(), (*guid)(), getsid(0), filename);

while (*ptr && size > 0)
    size -= sprintf((logstring+MAX-size), size, "
%s",&(*ptr++));

    syslog(LOG_INFO, "%s", logstring);
    free(logstring);
    closelog();
}

int execve(const char *filename, char **argv, char **envp) {
    static int (*func)(const char *, char **, char **);

    FN(func,int,"execve", (const char *, char **, char **));

#ifdef ROOT_ONLY
    if ((*guid)() != 0) return (*func) (filename, argv, envp);
#endif

    log(filename, argv);

    return (*func) (filename, argv, envp);
}

int execv(const char *filename, char **argv) {
    static int (*func)(const char *, char **);

    FN(func,int,"execv", (const char *, char **));

#ifdef ROOT_ONLY
    if ((*guid)() != 0) return (*func) (filename, argv);
#endif

    log(filename, argv);

    return (*func) (filename, argv);
}

```

E6) Database Creation SQL statements

```
CREATE TABLE users
(
    UserID MIDDLEINT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (UserID),
    loginname VARCHAR(20) NOT NULL,
    unixuid MIDDLEINT UNSIGNED,
    unixgid MIDDLEINT UNSIGNED,
    adsid VARCHAR(50),
    firstname VARCHAR(20),
    middlename VARCHAR(20),
    lastname VARCHAR(35),
    ADdomain VARCHAR(20),
    homeonhost VARCHAR(45) NOT NULL,
    Crole TINYINT UNSIGNED,
    Csysadm TINYINT UNSIGNED,
    Ccriticalfiles TINYINT UNSIGNED,
    Cphysicalaccess TINYINT UNSIGNED,
    Fattributes TINYINT UNSIGNED,
    allfortargetos CHAR(1),
    usercategoryflag VARCHAR(20),
);
```

```
CREATE TABLE hosts
(
    HostID MIDDLEINT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (HostID),
    hostip VARCHAR(50) NOT NULL,
    targetos VARCHAR(200) NOT NULL,
    signedir VARCHAR(200) NOT NULL,
    noofusers MIDDLEINT UNSIGNED NOT NULL,
    usrmd5sum VARCHAR(35) NOT NULL,
);
```

```
CREATE TABLE signatures
(
    SignID MIDDLEINT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (SignID),
    targetos VARCHAR(200) NOT NULL,
    hostip VARCHAR(50) NOT NULL,
    Crole TINYINT UNSIGNED NOT NULL,
    creationday TINYINT UNSIGNED NOT NULL,
    creationmonth TINYINT UNSIGNED NOT NULL,
    creationyear TINYINT UNSIGNED NOT NULL,
    signfile VARCHAR(200) NOT NULL,
    reason VARCHAR(30) NOT NULL,
    keyword1 VARCHAR(45) NOT NULL,
    keyword2 VARCHAR(45) NOT NULL,
    keyword3 VARCHAR(45) NOT NULL,
);
```

```
CREATE TABLE events
(
    EventID MIDDLEINT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (EventID),
    userid MIDDLEINT UNSIGNED NOT NULL,
    signid MIDDLEINT UNSIGNED NOT NULL,
    Fbreadth TINYINT UNSIGNED,
```

```

    Fappscore TINYINT UNSIGNED,
    SCPU TINYINT UNSIGNED,
    SRAM TINYINT UNSIGNED,
    SSIMAPPS TINYINT UNSIGNED,
    Fresutil TINYINT UNSIGNED,
    Fsophistication TINYINT UNSIGNED,
    Fexecops TINYINT UNSIGNED,
    Fnetops TINYINT UNSIGNED,
    Fbehavior TINYINT UNSIGNED,
    EPT TINYINT UNSIGNED,
);

```

E7)'hostregister' script

```

#!/usr/bin/perl -w

use strict;
use DBI;

#Essential Sanity checks...
my @whoami=getpwuid($<);
die "cmdregister Error:You should execute this program ONLY with root
privileges. You are not root.\n"
if ($whoami[2]!=0 && $whoami[3]!=0);

#Check that we can connect to the RDBMS server.
#We assume that the MySQL password has been configured in
#a way that allows not to use the -p option. In that way,
#we do not list the password in ths file, so that it is
#vulnerable and induce compile time dependencies. The
#standard way of achieving this is by having the usual
#/username/.my.cnf file containing the RDBMS server FQDN
#the username and the password.

die "registerhost Error:No MySQL client cnf file found.\n"
if (! (-e "/root/.my.cnf"));

#We hardwire the connection to the pre-release database here:
my
$datasource="DBI:mysql:itpmpreleasel;mysql_read_default_file=/root/.m
y.cnf";

my $itpmservh=DBI->connect ($datasource, undef, undef, {RaiseError =>
1, PrintError => 1});

#Do some essential data gathering on the local host.
#Note, what used to be localhostname var in version1beta
#has now switched to localip.
chomp(my $localhostname=`/bin/hostname`);
chomp(my $localtargetos=`/bin/uname -rsm`);
my $localsignaturedir="/var/log/signatures$localhostname";

print "Enter the IP address of the host: ";
chomp (my $localip=<STDIN>);
#debug

```

```

print "Checking registration details for host: $localhostname with IP
$localip \n";

#Does the signature directory exist?
if (!( -e $localsignaturedir)) {
    print "The directory $localsignaturedir does not exist.\n";
    print "Creating $localsignaturedir directory for you now. \n";
    chdir '/var/log/';
    mkdir "signatures$localhostname",0744;
    chdir "signatures$localhostname";
}

my @loginnames=`/bin/cat /etc/passwd | grep -v nologin | grep -v
"/bin/false"| grep -v ^halt | grep -v ^shutdown | grep -v ^nobody |
grep -v ^"sshd" | grep -v ^"uucp" | grep -v ^"bin" | grep -v ^"news"
| grep -v ^"sync" | grep -v ^"mail" | grep -v ^"root" | grep -v ^"lp"
| cut -d":" -f 1`;

#Caution here. If someone just adds an entry when we finish parsing
the loginnames and then do
#the md5sum, we might have a potential race hazard. Locking the
/etc/passwd file during this
#operation is a solution, but I won't just do that yet.
chomp(my $localusrmd5sum=`/usr/bin/md5sum /etc/passwd | cut -d" " -
f1`);
my $localnoofusers=$#loginnames;

#Debug:
print "Got $localnoofusers users with an md5sum of $localusrmd5sum
\n";

#my @unixuids=`cat /etc/passwd | grep -v nologin | grep -v
"/bin/false"| grep -v ^halt | grep -v ^shutdown | grep -v ^nobody |
grep -v ^"sshd" | grep -v ^"uucp" | grep -v ^"bin" | grep -v ^"news"
| grep -v ^"sync" | grep -v ^"mail" | grep -v ^"root" | grep -v ^"lp"
| cut -d":" -f 3`;

#my @unixgids=`cat /etc/passwd | grep -v nologin | grep -v
"/bin/false"| grep -v ^halt | grep -v ^shutdown | grep -v ^nobody |
grep -v ^"sshd" | grep -v ^"uucp" | grep -v ^"bin" | grep -v ^"news"
| grep -v ^"sync" | grep -v ^"mail" | grep -v ^"root" | grep -v ^"lp"
| cut -d":" -f 4`;

#Does a host entry exist in the itpmdb? If yes, the host is already
registered.
#but we might like to check the users (users might be added or
removed).
#If not, we register and record all the users by doing the following:
#
my $SQLh=$itpmservh->prepare("SELECT hostip FROM hosts WHERE
hostip='$localip'");
$SQLh->execute();

#Look up the list of hosts.
#A fetchrow loop is NOT necessary, since there should only be one
host entry in the database.
my @itpmhosts=$SQLh->fetchrow_array();
#If the database finds an entry, it will be the first element of the
array.

```

```

#If it does not find an entry, DBI returns empty strings so, we check
with
#undefs. In any other case, we bailout.
if ($itpmhosts[0] eq $localip) {
    $SQLh->finish(); alreadypresent();} elsif
(!defined($itpmhosts[0])) {
    doregister();} else { $SQLh->finish();    bailout(); }
    #print join ("\t", @itpmhosts), "\n";

#$itpmservh->disconnect();

#Subroutine definitions start here.

sub doregister {
    print "registerhost doregister: Host $localhostname with IP
$localip is not registered with the ITPM database. \n";
    print "registerhost doregister: Attempting to register
$localhostname . \n";
    #Get the host in the hosts table.
    my $rows=$itpmservh->do ("INSERT INTO hosts
(hostip,targetetos,signaturemdir,noofusers,usrmd5sum) "
        . " VALUES
('$localip','$localtargetetos','$localsignaturemdir',"
        . "'$localnoofusers','$localusrmd5sum')")
);
    if (($rows==1) || (!defined($rows))) {
        print "registerhost Fatal Error: doregister: No records
were altered in the hosts table. Host $localhostname was not
registered.\n";
    } else { print "registerhost doregister: The $localhostname
host with IP $localip was successfully registered in the database
hosts table. ! \n";}

    #And now register the users of the host.
    #Before we do that, the Application Logic dictates that we
    #need to investigate whether there are stale user entries on
    #the host table (if the host did not exist before, then it
would
    #impossible to have user entries on the users table). These
entries
    #will be removed from the users table.(the script was written
before INNODB
    #so this was the only way to enforce Referential Integrity,
since MySQL did
    #not support foreign keys).
    $rows=$itpmservh->do("DELETE FROM users WHERE
homeonhost='$localhostname'");

    #Now we are ready to start populating the users table.
    #Bogus values for uids and gids as they are given below.
    #We do this in order to trace bugs. Will have a unified loop
later.
    foreach my $usrforreg (@loginnames) {
        chomp $usrforreg;
        my $unixusid=getpwnam($usrforreg);
        my $unixgrid=getgrnam($usrforreg);

```

```

        $rows=$itpmservh->do("INSERT INTO users
(loginname,unixuid,unixgid,homeonhost) "
        . "VALUES
('$usrforreg','$unixusid','$unixgrid','$localhostname')");
    }
}

sub alreadypresent {
    #If already present, do we actually need to re-register the
users?
    print "registerhost alreadypresent: Host $localhostname with ip
$localip is already in the database. \n";
    print "registerhost alreadypresent: Checking if we need an
update on the users. \n";
    $SQLh=$itpmservh->prepare("SELECT usrmd5sum FROM hosts WHERE
hostip='$localip'");
    $SQLh->execute();
    my @fetchedchecksum=$SQLh->fetchrow_array();
    if ($fetchedchecksum[0] eq $localusrmd5sum) {
        print "registerhost alreadypresent: The host appears to
have the same users. No need for users update. \n";
        print "registerhost alreadypresent: So, host
$localhostname with IP $localip is already registered and requires no
updates. \n";
        $SQLh->finish();
    } elsif (!(defined($fetchedchecksum[0]))) {
        print "registerhost Fatal Error: alreadypresent:
usrmd5sum could not be retrieved. This is really bad!\n";
    } else {
        #If we need to re-register the users, then the simplest
thing is to re-register the
        #host. So, what we do is delete the hostname from the
hosts table and then call
        #again the doregister() subroutine. In this way, we re-
use the code and also accommodate
        #for the case that the host has been re-installed the OS
from scratch and/or stale entries.
        print "registerhost alreadypresent: We need to re-
register the users. \n";
        print "registerhost alreadypresent: Attempting to remove
the host with IP $localip. \n";
        my $rowsaffected=$itpmservh->do("DELETE FROM hosts WHERE
hostip='$localip'");

        print "registerhost alreadypresent: Calling doregister to
re-register the host from scratch. \n";
        doregister();
    }
}

sub bailout {
    print "registerhost Fatal Error: The database hosts table
returned an unexpected value! \n";
    print "registerhost Fatal Error: Value was : $itpmhosts[0] \n";
    $SQLh->finish();
    $itpmservh->disconnect();
}

```


References

- [1] National Computing Centre (1998), “*BISS '98. Information Security: The True Cost to Business. Research Report*”, The National Computing Centre Limited, UK.
- [2] Bace R. (2000), “*Intrusion Detection*”, First Edition, Macmillan Technical Publishing, Indianapolis, USA, pages:29-30.
- [3] Phoenix S. (1997), ‘*Cryptography, trusted third parties and escrow*’, BT Technology Journal, BT Laboratories, Suffolk, England, Volume 15, Number 2, April 1997
- [4] Skevington P., Hart T., ‘Trusted Parties in Electronic Commerce’, BT Technology Journal, BT Laboratories, Suffolk, England, Volume 15, Number 2, April 1997
- [5] Ko C., Frinkle T., Goan T., Heberlein T., Levitt K., Mukherjee B., Wee C (1993), ‘Analysis of an Algorithm for Distributed Recognition and Accountability’, Proceedings of the First ACM Conference on Computer and Communication Security, Fairfax, VA, pages:154-164.
- [6] Swicky E., Cooper S., Chapman B. (2000), ‘Building Internet Firewalls’, Second Edition, O’Reilly and Associates, Sebastopol, CA, ISBN: 1-56592-971-1: Chapter 4, page 99 describes the ‘land’-based attack amongst others.
- [7] Garfinkel S, Spafford G. (1996), ‘Practical UNIX and Internet Security’, Second Edition, O’Reilly and Associates, Sebastopol, CA, ISBN: 1-56592-148-1
- [8] Denning D.(1986), ‘An Intrusion Detection Model’, Proceedings of the Seventh IEEE Symposium on Security and Privacy, May 1986: pages 119-131.
- [9] Anderson, James P., ‘Computer Security Technology Planning Study 2. ESD-TR-73-51, Bedford, MA: Electronic Systems Division, Air Force Systems Command, Hanscom Field, October 1972.
- [10] National Computer Security Center, “A Guide to Understanding Audit in Trusted Systems”, NCSC-TG-001, July 28, 1987.
- [11] National Computer Security Center, “Department of Defense Trusted Computer System Evaluation Criteria.” Orange book, DOD 5200.28-std, December 1985.
- [12] Halme L., Lunt T., and Van Horne J, “Automated Analysis of Computer System Audit trails for Security Purposes.”, Proceedings of the National Computer Security Conference, Washington, D.C., September 1986.

- [13] Lunt, T et al., "IDES: A progress report", Proceedings of the Sixth Annual Computer Security Applications Conference, Tucson, AZ, December 1990.
- [14] Amoroso, E., "Intrusion Detection: An introduction to Internet surveillance, correlation, traps, trace-back, and response", Second Edition, Intrusion.Net books, NJ, ISBN:0-9666700-7-8, June 1999. Pages100-102 define the term 'intrusion' whereas page 16 contains the definition of 'Intrusion Detection'. Finally pages 27-28 contain comments related to the absence of GUI standards concerning the intuitive representation of intrusion related information in IDS designs.
- [15] Doyle J. Shrobe H, Szolovits P. (2000), 'On widening the Scope of Attack Recognition Languages', Massachusetts Institute of Technology, Cambridge, MA 02139
- [16] Papadaki M., Magklaras G., Furnel S., Alayed A. (2001), 'Security vulnerabilities and System Intrusions – The need for Automatic Response Frameworks', Proceedings of the 2001 Small Systems Security International Conference, Las Vegas, USA, September 2001.
- [17] Staniford-Chen, S. and Heberlein T., "Holding Intruders Accountable on the Internet", Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, CA, May 8-10, 1995.
- [18] Ranum M., Landfield Kent, Stolarchuk M., Sienkiewicz M., Lamberth A., Wall E., "Implementing a General Tool for Network Monitoring", paper available at <http://www.nfr.net/publications/LISA-97.html>
- [19] Schneier B. (2001), 'Managed Security Monitoring: Network Security for the 21st Century', Computers and Security Journal, Elsevier Science Ltd., Volume 20, pages 491-503.
- [20] Teng H., Chen K., Lu S. (1990), 'Security Audit Trail Analysis Using Intrusive Generated Predictive Rules', Proceedings of the 11th IEEE National Conference on Artificial Intelligence Applications, March 1990, pages 24-29.
- [21] Kumar S. (1995), 'Classification and Detection of Computer Intrusions', PhD Thesis, Purdue University, 1995.
- [22] Me, L. (1998), 'GASSATA, A Generic Algorithm as an Alternative Tool for Security Audit Trail Analysis', First International Workshop on the Recent Advances in Intrusion Detection, Louvian-la-Neuve, Belgium, September 1998.

- [23] Cohen F. (1994), 'A Short Course on Computer Viruses', John Wiley and Sons, ISBN:0-471-38367-8.
- [24] Goodrich M., Tamassia R. (2001), 'Data structures and Algorithms in Java', Second Edition, John Wiley and Sons Inc., ISBN:0-471-38367-8.
- [25] Porras P. (1992), 'A state transition analysis tool for Intrusion Detection', University of California, paper found at <http://www.csl.sri.com>
- [26] Kumar S., Spafford E. (1994), "A Pattern Matching Model for Misuse Intrusion Detection", The COAST Project, Department of Computer Sciences, Purdue University, USA
- [27] Newsham T., Ptacek T. (1998), "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", Technical Paper, Secure Networks Inc., URL: <http://www.securityfocus.com/library/745>
- [28] Smaha, S., "Haystack: An Intrusion Detection System", Proceedings of the Fourth IEEE Aerospace Computer Security Applications Conference, Orlando, FL, December 1988.
- [29] The Common Intrusion Detection Framework (CIDF), URL: <http://www.isi.edu/~brian/cidf>
- [30] The Intrusion Detection Working Group is hosted at the IETF's Working Group site, URL:<http://www.ietf.org/idwg>
- [31] Furnell S., Dowland P. (2000), 'A conceptual architecture for real-time intrusion monitoring', Information Management and Computer Security, MCB University Press, Volume 8 Number 2, pages 65-74.
- [32] Pfleeger C., Pfleeger S. (2003), 'Security in Computing', Third Edition, Prentice Hall, ISBN:0130355488.
- [33] Anderson J. (1980), 'Computer Security Threat Monitoring and Surveillance', James P. Anderson Co., April, Fort Washington, PA.
- [34] "SNORT: The Open Source Intrusion Detection System", URL: <http://www.snort.org/about.html>
- [35] Wu S., Manber U. (1994), "A Fast Algorithm For Multi-Pattern Searching", Department of Computer Science, University of Arizona.
- [36] SourceFire Incorporation (2003), "*High Performance Multi-Rule Inspection Engine* ", Vendor Paper outlining the SNORT Rules Definition Language , URL: <http://www.sourcefire.com>

- [37] SourceFire Incorporation (2003), “*Real Time Network Awareness™: Redefining the Intrusion Detection Industry*”, Vendor Paper outlining its latest IDS Commercial Framework solutions, URL: <http://www.sourcefire.com>
- [38] More information about the NID family of products can be found at the Vendor’s web site, URL: <http://www.nfr.com>
- [39] Subramanian M. (2000), “*Network Management: Principles and Practice*”, Addison Wesley, ISBN: 0201357429
- [40] Harrington D., Presuhn R., Wijnen B. (1998), “*An Architecture for Describing SNMP Management Frameworks*”, Request For Comments (RFC) 2271, Network Working Group
- [41] Waldbusser S. (1995), “*Remote Monitoring Management Information Base*”, Request For Comments (RFC) 1757, Network Working Group
- [42] Internet Security Systems (ISS) Vendor Web Site, URL: <http://www.iss.net/download/>
- [43] Gibson J., Huntington-Lee J., Terplan K. (1997), “*HP’s Open View*”, The McGraw-Hill Series on Computer Communications , ISBN: 0070313822
- [44] Network Magazine On-line Portal (2002), “Strategies & Issues: Thwarting Insider Attacks” , article written by Jim Carr, URL: <http://www.networkmagazine.com/article/NMG20020826S0011>
- [45] E-Trust software suite, Computer Associates Vendor Web Site, URL: <http://www3.ca.com/Solutions/Product.asp?ID=3224>
- [46] Palisade Systems VendorWebSite, URL:<http://www.palisesys.com/products/firemarshal/index.shtml>
- [47] Gibbons, R. (1992), “A Primer in Game Theory”, Prentice Hall International, ISBN: 0745011594
- [48] Helbig K. (1993), “Modelling the Earth for Oil Exploration: Final Report of the CEC’s Geoscience I Program 1990-1993”, Pergamon Publishing, ISBN: 0080424198
- [49] VNUnet Internet portal (2003), “Most cyber-attacks will come from insiders”, article written by Robert Jacques, URL: <http://www.vnunet.com/News/1141354>

- [50] Neumann P. (1999), 'The Challenges of Insider Misuse', SRI Computer Science Laboratory, Paper prepared for the Workshop on Preventing, Detecting, and Responding to Malicious Insider Misuse, 16-18 August 1999, RAND, Santa Monica, CA.
- [51] Caelli W., Longley D., Shain M. (1991), 'Information Security Handbook', Stockton Press.
- [52] PriceWaterhouseCoopers Internet portal (2004), "Information Security Breaches Survey 2004 – Technical Report", URL: http://www.pwc.com/images/gx/eng/about/svcs/grms/2004Technical_Report.pdf
- [53] Computer Security Institute (2003), "2003 CSI/FBI Computer Crime and Security Survey", URL: <http://www.gocsi.com/>
- [54] Computer Security Institute (2002), "2002 CSI/FBI Computer Crime and Security Survey", Computer Security Issues & Trends, Vol. VIII, NO.1.
- [55] Computer Security Institute (2001), "2001 CSI/FBI Computer Crime and Security Survey", Computer Security Issues & Trends, Vol. VII, NO.1.
- [56] TecSec Internet portal, URL: <http://tecsec.com/SetFrame.asp?Sec=CKM1>
- [57] Rapid 7 Internet portal, URL: <http://www.rapid7.com/docs/NetworkSecuritySurvey.pdf>
- [58] Computerworld Internet Portal (2000), "The Cyber-Mod Squad Sets Out After Crackers", Article written by Deborah Radcliff documenting the case of Abdelkader Smires
URL: <http://www.computerworld.com/news/2000/story/0,11280,45927,00.html>
- [59] ZDnet Internet Portal (2001), "Firms shop around for Net law jurisdictions", article written by Wendy McAuliffe
URL: <http://news.zdnet.co.uk/business/0,39020645,2085983,00.htm>
- [60] The University of Oslo November 2002 widespread cracking incident was quoted amongst various sources in the Cybersecurity- Infrastructure protection website of Dartmouth College, USA.
URL: <http://news.ists.dartmouth.edu/snms/1102.htm#30>
- [61] PC World Internet portal (2001), "Are employees wasting time on-line?" article written by Scarlet Pruitt, URL: <http://www.pcworld.com/news/article/0,aid,56947,00.asp>

- [62] Indiana University Unix System Support Group (1997), "Unix Workstation System Administration Education Certification Course", On-line manual in URL: <http://www.ussg.iu.edu/edcert/session2/disk/diskspace.html>
- [63] O-Reilly Net Portal (2001), "System-and Network-wide bandwidth shaping for P2P apps", article written by Damien Stolarz, URL: <http://www.oreillynet.com/pub/wlg/549>
- [64] Bishop M. (1995), "Intruders and UNIX", Technical Presentation, Department of Computer Science, University of California at Davis, slide 6 mentions the relevant intrusion concealment tools
URL: <http://seclab.cs.ucdavis.edu/projects/vulnerabilities/scriv/1995-ns.pdf>
- [65] Furnell S., Magklaras G., Papadaki M., Dowland P. (2001), 'A Generic Taxonomy for Intrusion Specification and Response', Proceedings of Euromedia 2001, Valencia, Spain, pages: 125-131.
- [66] Neumann P., Parker D. (1989), 'A summary of computer misuse techniques', In Proceedings of the 12th National Computer Security Conference, Baltimore, USA, pages: 396-407.
- [67] Lindqvist U., Jonsson E. (1997), "How to systematically classify Computer Security Intrusions", Proceedings of the 1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, IEEE Computer Society Press.
- [68] Howard, J. (1997), "An Analysis of Security Incidents on the Internet 1989-1995", PhD Thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.
- [69] Tuglular T. (2000), "A preliminary Structural Approach to Insider Computer Misuse Incidents", EICAR 2000 Best Paper Proceedings: pages 105-125.
- [70] Magklaras G., Furnell S. (2002), "Insider Threat Prediction Tool: Evaluating the probability of IT misuse", Computers & Security, Elsevier Science Ltd, Vol. 21, No. 1, pages: 62-73.
- [71] Shaw E.D., Ruby K.G., Post J.M. (1998), "The Insider Threat to Information Systems", Security Awareness Bulletin, No. 2/98, Political Psychology Associates Ltd.
- [72] Bach M. (1986), 'The design of the UNIX Operating System', Prentice Hall International Editions, NJ, 1986.
- [73] Richter J. (1997), 'Advanced Windows', Microsoft Press, Redmond, Washington.

- [74] Aslam T., Krsul I., Spafford E., 'Use of a Taxonomy of Security Faults', Technical Report TR-96-051, COAST Laboratory, Department of Computer Sciences, Purdue University, IN, 1996.
- [75] Moore D., Voelker G., Savage S. (2001), "Inferring Internet Denial of Service Activity", Proceedings of the 2001 USENIX Security Symposium, Washington D.C.
- [76] Frykholm N., (2000), "Countermeasures against Buffer Overflow Attacks", White Paper, RSA Laboratories.
- [77] Postel J., Reynolds J. (1983), "TELNET PROTOCOL SPECIFICATION", IETF Network Working Group Internet Request For Comments (RFC) Number 854.
- [78] Ylonen T. (1995), "The SSH (Secure Shell) Remote Login Protocol", IETF Network Working Group Internet Draft, Helsinki University of Technology, URL: <http://www.free.lp.se/fish/rfc.txt>
- [79] Ziegler R. (2002), "LINUX Firewalls", Second Edition, New Riders Publishing, ISBN: 0735710996, Chapter 2, pages 48-50.
- [80] Sharda N. (1999), "Multimedia Information Networking", Prentice Hall Inc., ISBN: 0132587734, Chapter 12.
- [81] Sommerville I. (2000), "Software Engineering", Addison Wesley, ISBN: 020139815X
- [82] Wood B. (2000). "An insider threat Model for Adversary Simulation", SRI International, Research on Mitigating the Insider Threat to Information Systems - #2: Proceedings of a Workshop Held by RAND, August 2000.
- [83] Schultz, E.E. (2002). "A framework for understanding and predicting insider attacks", Computers & Security, vol. 21, no. 6, pp. 526-531.
- [84] Dubois P. (2003). "MySQL™, The definitive guide to using, programming and administering MySQL 4 databases", New Riders Publications, ISBN: 0735712123.
- [85] Evans, G., Simkin, M. 1989. "What Best predicts Computer Proficiency?" Communications of the ACM (32:11), November 1989, pages 1322-1327.
- [86] Huff S., Munro M., Marcolin B. 1992. "Modeling and measuring End User Sophistication", University of Western Ontario, Canada, Paper Published on the 1992 ACM Proceedings, ACM 089791-501-1/92/0004/0001

- [87] EMBOSS.org portal, The European Molecular Biology Open Software Suite.
<http://www.emboss.org>
- [88] The Basic Local Alignment Search Tool (BLAST).
<http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/references.html>
- [89] Socolofsky T., Cale C. (1991) "A TCP/IP Tutorial", Internet Request For Comment (RFC) number 1180, Network Working Group, Internet Engineering Task Force.
- [90] International Standards Organisation (ISO) web portal. "Open systems interconnection in general", ICS Field 35.100.01.
- [91] Kapoor A. (1992). "SNA: Architecture, Protocols and Implementation", McGraw-Hill Education, ISBN: 0070337276
- [92] Schneeweiss W. (1989). "Boolean Functions: With Engineering Applications and Computer Programs", Springer Verlag Publications, ISBN: 0387188924.
- [93] Lee, A. (1992). "Investigations into history tools for user support." Ph.D. Thesis, Department of Computer Science, University of Toronto, Ontario, Canada.
- [94] Greenberg, S. (1993). "The computer user as toolsmith: The use, reuse, and organization of computer-based tools". Cambridge University Press
- [95] Davison B., Hirsh H. (1998). "Predicting Sequences of User Actions", Working Notes of the Joint Workshop on Predicting the Future: AI Approaches to Time Series Analysis, Fifteenth National Conference on Artificial Intelligence (AAAI98)/Fifteenth International Conference on Machine Learning (ICML98).
- [96] Brassard G. (1985). "Crusade for a better notation", SIGACT News, vol.17, no. 1, pages 60-64.
- [97] Cybercast News Service portal. "Anti-Porn Bill Targets Internet 'File Sharing'", article written by Lawrence Morahan, CNSNews.com Senior Staff Writer, July 28 2003.
URL:<http://www.cnsnews.com/ViewNation.asp?Page=%5CNation%5Carchive%5C200307%5CNAT20030728a.html>
- [98] The Mutella web portal software can be found on the following URL:
<http://mutella.sourceforge.net/>
- [99] Netscape Communications Web Portal. "Viewing or clearing the Netscape History File",
URL:

- <http://help.netscape.com/kb/consumer/19960627-14.html>
- [100] ZDNET UK Internet Portal . "Supercomputers to run Windows", article written by Stephen Shankland and Ina Fried, May 25th 2003. URL:<http://insight.zdnet.co.uk/hardware/servers/0,39020445,39155685,00.htm>
- [101] The LINUX Operating System Internet portal. URL: <http://www.linux.org>
- [102] The FreeBSD Operating System Internet portal. URL: <http://www.freebsd.org>
- [103] Tech Web Internet Portal. "Linux Paces Strong Server Growth", article written by W. David Gardner, May 28th 2004, URL:
<http://www.techweb.com/wire/story/TWB20040528S0007>
- [104] PC World Internet portal. "Will Your Next Desktop PC Run Linux? The alternative OS is finally gaining momentum on the desktop.", article written by Alexandra Krasne, May 27th 2004. URL:<http://www.pcworld.com/news/article/0,aid,116298,00.asp>
- [105] Ousterhout J. (1998). "Scripting: Higher Level Programming for the 21st Century", IEEE Computer magazine, March 1998 issue, pages pp. 23-30.
- [106] "The Great Computer Language Shootout", Doug Bagleys programming language performance guide, regular expression section. Note: This informal benchmark guide quotes results that are based on the performance of compilers on the year 2001. URL:
<http://www.bagley.org/~doug/shootout/bench/regexmatch/>
- [107] Microsoft TechNet Internet portal. URL:
http://msdn.microsoft.com/library/default.asp?url=/library/enus/adschema/adschema/a_securityidentifier.asp
- [108] The PostgreSQL RDBMS Internet portal. URL: <http://www.postgresql.org/>
- [109] Fermi National Accelerator Laboratory Internet portal. "PostgreSQL or MySQL?", PostgreSQL-MySQL RDBMS comparison guide written by the Laboratory's Database Systems Group, URL:
<http://www-css.fnal.gov/dsg/external/freeware/pgsql-vs-mysql.html>
- [110] Oracle Corporation Internet portal. URL: <http://www.oracle.com/products/>
- [111] Sybase Corporation Internet portal. URL: <http://www.sybase.com/home>
- [112] International Business Machine Corporation DB2 Internet portal. URL:
<http://www-306.ibm.com/software/data/db2/>

- [113] Infoworld Internet portal. "Affordable 64-bit computing", article written by Tom Yager, June 18th 2004. URL:http://www.infoworld.com/article/04/06/18/25FE64bits_1.html
- [114] Dwivedi H. (2003). "Implementing SSH: Strategies for Optimizing the Secure Shell", John Wiley & Sons INC, ISBN: 0471458805.
- [115] U.S. Department of Commerce/National Institute of Standards and Technology.(1999). "Data Encryption Standard (DES)", FIPS PUB 46-3.
- [116] U.S. Department of Commerce/National Institute of Standards and Technology.(2001). "Advanced Encryption Standard (AES)", FIPS PUB 197.
- [117] Schneier B.(1993). "Fast Software Encryption", Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.
- [118] Diffie W., Hellman M.(1976). "New directions in cryptography", IEEE Transactions on Information Theory, Volume 22, pages 644-654.
- [119] Diffie W., Van Oorschot, Wiener M. (1992), "Authentication and authenticated key exchanges", Designs, Codes and Cryptography Journal, Volume 2, pages 107-125.
- [120] Leach P., Perry D. (1996). "CIFS: A Common Internet File System", Microsoft Internet Developer periodical, November 1996 issue.
- [121] Lechnyr D. (2004). "The Unofficial Samba HOWTO", The 'Introduction' section contains a reference to a document with title "CIFS:Common Vulnerabilities Fail Scrutiny", written in 1997, and discussing a range of data security issues of the CIFS protocol at a technical level.
- [122] SAMBA" open source project Internet portal. URL: <http://www.samba.org>
- [123] Johnson M., Troan E. (1998). "LINUX Application Development", Addison Wesley, ISBN: 0201308215
- [124] "Snoopy" logger project Internet portal. URL: <http://sourceforge.net/projects/snoopylogger/>
- [125] Mano M. (1993). "Computer Systems Architecture", International Edition, Prentice Hall, ISBN: 0131757385
- [126] Liu J., Balasubramanian Chandrasekaran Yu W., Sushmitha P., Wyckoff P. (2004), "Microbenchmark performance comparison of high-speed cluster interconnects", IEEE Micro, Volume: 24, Issue: 1, On page(s): 42- 51
- [127] The Condor Project Home Page at the University of Wisconsin . URL: <http://www.cs.wisc.edu/condor/>

- [128] U.S. National Security Telecommunications And Information Systems Security Committee (1999), "The Insider Threat To US Government Information Systems", NSTISSAM INFOSEC /1-99.
- [129] Kaspersky Laboratories Internet Portal. URL: <http://www.kaspersky.com/technologies>
- [130] Symantec Internet portal. URL: <http://www.symantec.com/avcenter/reference/heuristc.pdf>
- [131] Magklaras G., Furnell S. (2004), "An End User Sophistication Model for Insider Threat Prediction in IT Systems", submitted to Computers and Security
- [132] Vacca J. (2002), "Computer Forensics: Computer Crime Scene", Charles River Media, ISBN: 1584500182
- [133] Ramming C. (1997), 'USENIX Conference on Domain Specific Languages', USENIX Association, Santa Monica, CA, USA
- [134] Feiertag R., Kahn C., Porras P., Schnackenberg D., Staniford-Chen S., Tung B. (1999), "A Common Intrusion Specification Language (CISL)", June 1999 revision, URL: <http://www.isi.edu/~brian/cidf/drafts/language.txt>
- [135] Doyle J. (1999), "Some representational limitations of the Common Intrusion Specification Language", Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, November 1999 Revision
- [136] Curry D., Debar H., Feinstein B. (2004), "The Intrusion Detection Message Exchange Format", Internet Draft, Intrusion Detection Exchange Format working group, Internet Engineering Task Force, URL: <http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-11.txt>
- [137] Feinstein B., Matthews G., White J. (2002), "The Intrusion Detection Exchange Protocol (IDXP)", Internet Draft, Intrusion Detection Exchange Format working group, Internet Engineering Task Force
URL: <http://www.ietf.org/internet-drafts/draft-ietf-idwg-beep-idxp-07.txt>
- [138] World Wide Web Consortium (W3C) Internet portal, "Extensible Markup Language", URL: <http://www.w3.org/XML/>